Computational analysis of a theorised tendon-based trunk support device

Investigating safety, predictability, and mechanical requirements

by

Spyros Leonard Lian Krinis

Student Number 5369134

Assessment Committee:

Supervisor and Chair:	Dr. Ing. L. Marchal-Crespo
Supervisor:	PhD R.F. Pangalila
Supervisor:	Dr. Ir. A.H. Stienen
External member:	Dr. Ir. G. Smit

Faculty of Mechanical Engineering (ME), Delft University of Technology

Contents

Abstract

Introduction: Movement disorders, such as Cerebral Palsy, prevent children from fully developing their motor coordination abilities reducing their quality of life. Trunk control is a core coordination competency but there is limited variety in sitting based therapies. This paper investigates a theorised trunk support device for children which uses a system of elastic ropes and pulleys to support and prevent them from falling whilst they perform trunk control training exercises in a seated position. The aim of this study is to understand the mechanical properties required to ensure this device behaves safely and predictably. The device is considered safe if it prevents the child from falling and predictable if it always pulls the child towards an upright seating position. Additionally, it was investigated if the mechanical properties could be satisfied by off-the-shelf components and two possible pulley systems were compared.

Methods: To theoretically test if this device concept could meet the three criteria, a MATLAB simulation was written. A genetic optimization was then used to find the least stringent mechanical properties that still satisfy the three criteria. Off-the-shelf components that satisfy the mechanical properties were then found.

Results: Both pulley systems satisfied the safety and predictability criteria. Moreover, the necessary mechanical properties could be satisfied by off-the-shelf components, though some design modifications may be required. Ultimately, the configuration involving shorter ropes was determined to be the better option as it had less stringent mechanical requirements and potential design modifications would likely be easier to implement.

Conclusion: This study concluded that the optimised device, theoretically, shows good performance and seems practical to build. Future work is required to design a harness that would allow a child to comfortably train with the device.

1 Introduction

1.1 Movement Disorders in Children

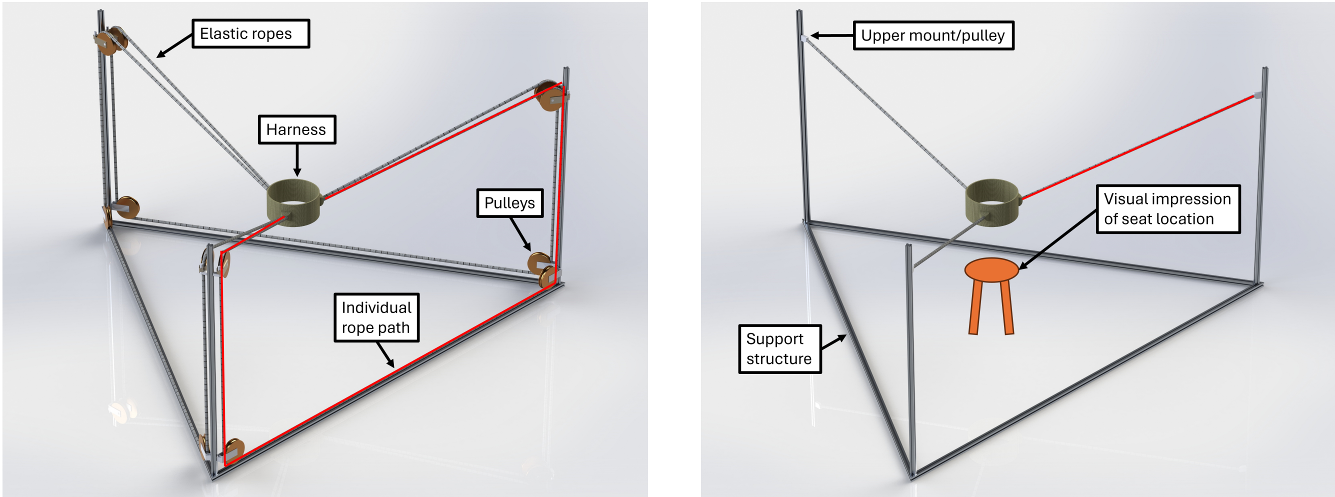
Cerebral Palsy is the most common cause of movement disorders and affects between two and three out of 1000 births. This figure can increase to 40–100 per 1000 premature births in babies born with low birth weight [1]. Whilst Cerebral Palsy is a developmental condition, genetic diseases such as Down's Syndrome and trauma such as a spinal cord injury can also weaken muscle function. Many of these conditions are chronic and their symptoms can only be alleviated but generally not corrected completely [2]. Symptoms, however, can be improved through physical training to improve muscle function [3].

1.2 Current Training Protocols

Therapy aims to improve the child's muscle coordination, strength, and endurance. The broad goal of therapy programs is to allow the child to perform key functional abilities including sitting upright, standing, and walking [2]. Before being able to stand or walk, the first key functional ability is sitting upright without support [4,5]. To achieve this, children must have control over their trunk muscles to be able to orient and balance themselves in this posture [6]. Currently, there are four broad training therapy techniques that target trunk control [2,3,7].

The first therapy technique for training trunk control is to use physical exercises to engage trunk muscles to improve strength and coordination. One example is having a child sit whilst a therapist asks them to lean or move to catch a ball. During this exercise, another therapist supports the child with their hands where necessary [8,9]. Another option is to use suit-and-cage therapy where the child wears a suit with a variety of mounting points. These mounting points are attached with elastic ropes to a large cage that partially surrounds the child. The ropes are configured by the therapist to support the child during training whilst the therapist engages with the child [10–13]. Alternatively, hippotherapy involves children riding horses and relies on the rhythmic motion of the horse's walking pattern as an external stimulus to train trunk muscles. Robotic devices are slowly also penetrating the field of trunk control rehabilitation. A robotic imitation of hippotherapy is available [14–16] as well as a novel technique called Trunk-Support-Trainer (TruST). With the TruST device, a patient sits in the center of the device and wears a harness that is attached to a set of stiff ropes guided by a pulley system. These ropes are actuated by electric motors to provide either assistive or resistive forces to allow for a more dynamic training session and provide a greater challenge or more support depending on the needs of the patient [17,18].

The theorised device analysed in this study targets a gap in the market between the suit-and-cage therapy and the TruST robotic system. The device has a similar support strategy as the TruST device, where the patient is seated and is supported by ropes guided by a pulley system, but uses a passive system based on elastic ropes similar to the suit-and-cage equipment. Using a completely passive system is expected to result in the theorised device being cheaper than the TruST system. Additionally, the suit-and-cage method has many different elastic elements that must be adjusted to each patient. Whilst this makes the system highly adaptable, adjusting these elements for each child can be time consuming leading to therapists having less time to train with the child. Since the support system is predefined, this restricts adjustments to only certain parameters in the case of the theorised device. This is expected to



(a) Three interconnected ropes configuration

(b) Three separate ropes configuration

Figure 1: Three-dimension models of the two tested rope configurations. Both images used parts similar to the off-the-shelf components that were found to suit the mechanical requirements found by this paper. These parts can be found in Appendix E.3.

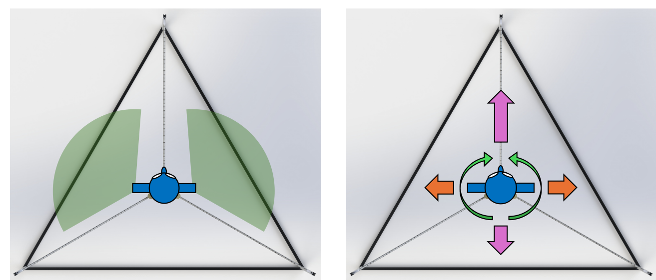
improve the speed adjustments can be made.

1.3 Device Concept

The device concept proposed in this paper involves supporting a child using a system of elastic ropes and pulleys. The child would be seated in the center of the device and would wear a harness to which the elastic ropes could attach to (see Figure 1 for the two configurations). The aim of this device is to support the trunk of a child, preventing them from falling, whilst they conduct trunk training exercises led by a therapist.

The main support structure of the device is made up of three evenly spaced columns that surround the child. With one column directly in front of the child, a three-column design allows for a nearly 120° region between ropes within which the child could move their arms freely (see Figure 2a). This allows the device to interfere as little as possible with the child's exercises.

There were two rope configurations theorised during the conceptualization of the device. The two options are: i) Three interconnected ropes, and ii) Three separate ropes configurations. The three interconnected ropes configuration involves having both ends of the rope connected to a mounting point on the harness. This is done by routing each rope through a set of pulleys that take the rope from one harness mounting point to another (see Figure 1a). The three separate ropes configuration involves one end of each rope being connected to the harness and the other end connected to a mount, rather than a pulley, at the top of the column (see Figure 1b). Though more rope configurations may exist, these options offered two highly plausible configurations.



(a) The green arcs indicate nearly 120° regions within which the child's arms could move freely without contacting any ropes.

(b) The three colors of arrows indicate the three degrees of freedom the patient can move their torso in whilst staying seated in the device.

Figure 2: Top down view of the theorised device highlighting different considerations of the design.

1.4 Device Requirements

To allow children to conduct physical therapy exercises as freely as possible, the device is designed to allow children to lean forward up to 40°, backward 22.5°, and medio-laterally 22.5°. The design also allows a child to twist their torso up to 45°. These degrees of freedom represent the safe workspace in which a child can safely move within when in the device (see Figure 2b for representation of the degrees of freedom and Table 2 for the range of the workspace). It was difficult to determine an exact workspace to target as each patient can have very different abilities. However, after consulting with an expert in the field, it was estimated that these values would provide a more than adequate range of motion for children to train within.

The device was designed for children between the ages of three and eight-years-old as these younger patients are better able to develop further than older ones [19]. Additionally, though suit-and-cage therapy is primarily for younger children, the TruST system is targeted towards adults hence leaving room in the market for a child focused tendon based device. However, as children within these ages can vary greatly in body proportions and weight, upper and lower limits were selected. Hence, the smallest and lightest child was selected to be a three-year-old female in the lowest 5th percentile of children. Conversely, the largest and heaviest child was an eight-year-old male in the highest 95th percentile of children. The specific values for the different anthropometric measures are listed in Table 1 and are sourced from the DINED Anthropometric database [20]. Only data for Dutch children was contained in this database hence only this population has been considered.

Anthropometric Measure	3YO Female 5th Percentile	8YO Male 95th Percentile	Unit
Standing Height	0.93	1.42	m
Weight	14	35	kg
Waist Circumference	0.43	0.81	m
Arm Length	0.66	1.04	m
Sitting Head Height	0.53	0.75	m
Sitting Shoulder Height	0.32	0.47	m
Seat Height (Knees at 90°)	0.22	0.40	m

Table 1: Anthropometry data for the two children representing the smallest and largest children included in the design of the trunk support device [20].

To accommodate for different sizes of child, an adjustable seat and harness are needed, however, both the seat and harness are considered outside the scope of this paper due to their complexity. For this paper, the seat is assumed to be adjustable in height to allow any child to sit comfortably in an upright position with their knees at 90° and their feet placed flat on the floor. The harness is assumed to be cylindrical in shape with three evenly spaced mounting points around its outer circumference (see Figure 1). This harness design would allow it to be attached at different heights along the child's torso. However, to focus the analysis of the paper, it was assumed that a harness height of 75% along the child's torso would be approximately where the harness would most likely be mounted (effect of harness height is investigated in Appendix C).

What is within the scope is alteration of the stiffness of the ropes, the pre-tensioning within the ropes, and the upper pulley/mounting offset. These three variables impact how the device reacts to the position of the child and are the primary variables that will be investigated within this paper. The ranges for rope stiffness, the pre-tensioning within the ropes, and the upper pulley/mounting offset were determined by estimating maximum and minimum values that provided a broad range to analyse (see Table 2).

Condition	Minimum	Maximum	Unit
Safe Workspace			
Medio-lateral Lean	± 22.5		°
Anterior-posterior Lean	22.5 (post)	40 (ant)	°
Twist	± 45		°
Range of Mechanical Properties			
Rope Stiffness	1	2000	N/m
Pre-tension	0	200	N
Upper Pulley/-Mounting Offset	-0.1	0.7	m

Table 2: Boundaries of the workspace the child can move within and key mechanical properties that can be manipulated to adjust the behaviour of the device.

In order to avoid children getting their fingers caught in the pulley mechanisms, the distance of the columns from the child must be determined. This was done by using Equation 1 which determines the maximum horizontal distance the largest child could stretch their hands whilst staying within the workspace.

$$R_{safe} = L_t \cdot \sin 40 + R_t \cdot \cos 45 + L_a + 0.1 \qquad (1)$$

40° and 45° correspond to the largest leaning forward and twisting amount allowed within the workspace. L_t is the length of their torso, R_t is the radius of their torso used in this case as the breadth of their shoulders, and L_a is the length of their arm. Finally, a safety margin of 0.1 m is added and the result is rounded up resulting in a distance of 1.54 m (Figure 3 represents the variables used to calculate the safe radius). This value is used as the distance the columns radiate out from the center of the child's torso in an upright sitting position.

1.4.1 Defining Safe and Predictable Device Behaviour

Broadly, the device is considered safe and predictable when it always provides a force that moves the child towards an upright seating position. Furthermore, the device is safe when the child is also unable to overpower the device and push beyond the safe workspace. Data

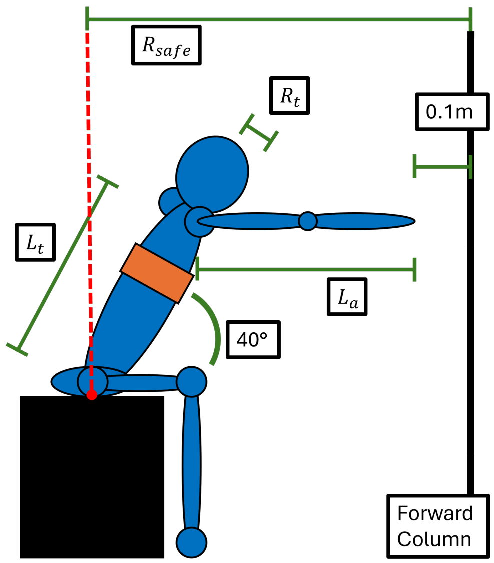


Figure 3: The different measurements used to determine the minimum safe radius from the center of the child's torso to the forward column. One variable that is not visible is the twist angle of 45° which corresponds to the twist of the torso. See equation 1 for how these factors are combined.

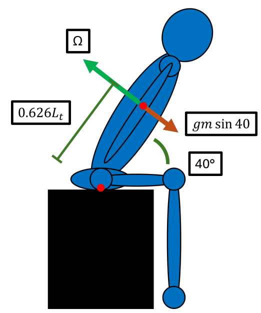
on the amount of force children can generate with their torso muscles was lacking, hence, it was estimated that the force children could generate is proportional to the weight of their upper bodies (See Equation 2).

$$\Omega = 2(0.65mg \cdot \sin 40 \cdot 0.626L_t) \tag{2}$$

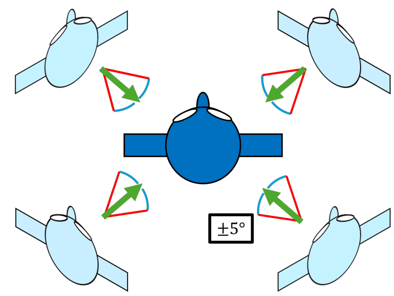
This strength value is referred to as "opposition strength" (Ω) and is based upon how much torque a child must generate to statically support their body when leaning forward at 40° multiplied by two[1]. m is the total mass of the child, g is the acceleration of gravity ($9.81 m/s^2$), and L_t is the length of their torso. Multiplying by two serves as a safety factor, 0.65 represents the percentage the upper-body makes up of the total weight, 40° represents the child leaning forward to the edge of the workspace, and 0.626 represents the percentage up the child's torso at which the center of mass is located for the child's upper body. The percentage of the total mass the upper-body makes up is based on information for able-bodied children [21]. The location estimate of the upper body's center of mass is based on the location of an adult upper body's center of mass as this information could not be found for children [22]. Figure 4a shows the different anthropometric measures used to calculate Ω.

The device is considered predictable when, at every location, the child is pulled towards an upright seating position. To allow for some margin of error, the motion of the child toward the upright position should not deviate more than 5° at any location. A representation of this can be found in Figure 4b.

[1] Ω represents a torque as the child's torso is modeled as an inverse pendulum as explained in section 2.1.



(a) Opposition strength anthropometric variables and forces.



(b) Looking from above, this shows how a safe and predictable device will always bring the child towards an upright seating position.

Figure 4: Representations of safety and predictability criteria for the device.

1.5 Research Question

In this paper, a solution that lies between the suit-and-cage-based therapy and the TruST solution is theorised, designed, optimized and evaluated through simulations. As this is the first work analysing this kind of passive device, fundamental questions about the performance and physical feasibility of the device must be answered. Namely, what are the mechanical requirements which ensure the safe and predictable behaviour of a novel, passive, tendon-based trunk support device designed for children with movement disorders? This question results in two further sub-questions. The first asks can off-the-shelf components be found that satisfy the mechanical requirements? The second compares two possible rope configurations to determine which one offers a safer and more predictable device that has the least rigorous mechanical requirements. To answer these questions, a MATLAB simulation of the device was written to understand how the device would react to the child leaning at different positions. To determine which mechanical properties to use for the two configurations, a genetic optimization was written to determine the least demanding mechanical properties that still satisfy the safety and predictability criteria.

2 Methods

To determine if the safety and predictability of the device can be achieved across the range of included child ages, mechanical properties for both the three-year-old (5^{th} percentile female) and eight-year-old (95^{th} percentile male) children must be identified for both rope configurations. Consequently, this results in four scenarios to test: i) the three-year-old in the interconnected ropes configuration, ii) the three-year-old in the separate ropes configuration, iii) the eight-year-old in the interconnected ropes configuration, and iv) the eight-year-old in the separate ropes configuration.

Code was written to analyse these four scenarios and involves three key components; the model of the child, the simulation of the device, and an optimization algorithm to determine mechanical properties for each scenario that satisfy the safety and predictability criteria. A key metric analysed to understand if the device has satisfied the criteria is the net torque vector. The net torque vector is a result of all the forces generated by the device acting on the child as well as the force of gravity acting on the child. The location these forces act upon the child's torso are taken into account thus resulting in a vector (process explained in section 2.2). This metric is used to understand the direction the child would move at any location within the workspace. This code quantifies safety and predictability with two criteria. The device is safe when the net torque acting on the child along the boundary of the workspace never falls below 110% of the child's opposition strength. The device is predictable when net torque at all locations directs the child within 5° of a direct path to the upright seating position.

2.1 Child Model

Each child is modeled using the Patient class (see Appendix F.5). This class takes the inputs of age, height, weight, waist circumference, and opposition strength. Age is used to match the rest of the child's anthropometric measures found in Table 1. The torso of the child is modeled as a rigid inverse pendulum with the mass of the child's upper body, head, and arms modeled as a point mass. This point mass is estimated to be 62.6% along their torso's length and 65% of the total body weight explained in section 1.4.1. This inverse pendulum moves about a fixed point located on the surface of the chair and allows for three degrees of freedom. This approach allows the modeled child to lean anterior-posteriorly and medio-laterally and twist their torso. Though this method does simplify the motion of the torso, it provides a base from which to understand how the device reacts to the child leaning with certain postures.

2.2 Device Simulation

The function "sim_device" (see Appendix F.2) is the component of the code that simulates the device. The code takes key mechanical properties of the device, rope configuration being simulated, child being modeled, and lean/twist information as input in order to simulate the device. With these inputs, the simulation of the device can happen in seven steps (see Appendix A for a detailed explanation).

The first step involves determining the locations of key points in three-dimensional space using anthropometry data of the modeled child (see Table 1) and the mechanical parameters of distance from the child to the support columns, upper pulley/mount offset, and rope configuration. Key points include the location of the columns, pulleys, harness mounting points, and the child's center of mass and center of rotation. With spatial information determined, information about the ropes can be calculated in step two with Equations 3 and 4.

$$l_\Delta = \frac{T_0}{K} \qquad (3)$$

$$L_u = L_0 - l_\Delta \qquad (4)$$

The stretched rope length is found by calculating the distance of the path taken by each rope (L_0) (see red lines in Figure 1). Using the given pre-tension value (T_0) and rope stiffness (K) and assuming the ropes are linearly elastic materials, the elongation length (l_Δ) is calculated using Hooke's Law (see Equation 3) [23]. After this, subtracting l_Δ from L_0 will give the unstretched rope length (L_u)(see Equation 4). From here, step three involves rotating the child and harness points, based on the given lean and twist inputs, around the center of rotation. A three-dimensional rotation matrix in XYZ order was used as this seemed to generate more realistic lean locations (see Appendix A). For step four, the new harness locations are used to determine the new lengths of each rope. This is then used to calculate how much tension is within the ropes with Hooke's Law. Then, the new tension within each rope (T_1) can be used to calculate the net force vector acting on each harness mounting point ($\vec{F}_h$) using Equations 5 and 6.

$$\hat{f}_h = \frac{\vec{P}_s - \vec{P}_h}{\|\vec{P}_s - \vec{P}_h\|} \qquad (5)$$

$$\vec{F}_h = \sum_{i=1}^{n} T_{1,i} \cdot \hat{f}_{h,i} \qquad (6)$$

For each rope, the unit vector of the displacement vector between a harness point ($\vec{P}_h$) and the corresponding upper pulley/mount location ($\vec{P}_s$) is calculated ($\hat{f}_h$). This will give the direction in which the tension in the rope is acting. This unit vector is then multiplied by T_1 to get the force vector from that rope acting on the harness mount. If there are multiple ropes attached to a harness point, the force vector for each rope is summed to get the net force vector ($\vec{F}_\tau$). Additionally, the force vector of gravity acting on the child's center of mass is also calculated based on the weight of the child's upper body. With $\vec{F}_\tau$ acting on each harness mounting point and gravity acting on the child's torso known, they can be decomposed into trunk-parallel forces ($\vec{F}_p$) and torque-generating forces ($\vec{F}_\tau$) using Equations 7 and 8.

$$\vec{F}_p = \frac{\vec{F}_h \cdot \vec{V}_t}{\|\vec{V}_t\|^2} \cdot \vec{V}_t \qquad (7)$$

$$\vec{F}_\tau = \vec{F}_h - \vec{F}_p \qquad (8)$$

To find $\vec{F}_p$, the projection of $\vec{F}_h$ onto the vector parallel to the trunk ($\vec{V}_t$) is used. $\vec{V}_t$ is taken as the displacement vector between the center of rotation and the center of mass of the child when leaning. The trunk-parallel forces at the harness points are used as a mechanical requirement that a future harness design must be able to sustain. $\vec{F}_\tau$ is found by getting the remainder of the force vector acting on the harness point from $\vec{F}_p$ with Equation 8. Finally, the net torque vector acting on the child ($\vec{\tau}_{net}$) can be calculated with Equation 9.

$$\vec{\tau}_{net} = L_h \cdot (\vec{F}_{\tau,l} + \vec{F}_{\tau,r} + \vec{F}_{\tau,f}) + 0.626 L_t \cdot \vec{F}_{\tau,g} \qquad (9)$$

First, the net torque-generating force on the harness is found by summing the torque-generating components acting at each harness point ($\vec{F}_{\tau,l}$, $\vec{F}_{\tau,r}$, $\vec{F}_{\tau,f}$ for the left, right, and front harness forces respectively). This is then multiplied by the length from the center of rotation to a harness point (L_h) to find the torque they collectively generate. Then, the torque generated by gravity is found by multiplying the length between the center of mass and center of rotation ($0.626L_t$, seen section 1.4.1) to the torque-generating component of the gravity force ($\vec{F}_{\tau,g}$). Then, these two products can be summed to find the net torque acting on the child ($\vec{\tau}_{net}$). The net torque vector is the main metric used to understand the behaviour of the device by indicating which direction the child would move at any given lean and twist posture. Note that the net torque vector does not follow the standard right-hand rule for denoting torques [24]. Instead, the net torque vector aligns with the direction of the force vector acting on the child's torso, but the net torque vector's magnitude still represents the net torque (see Figure 5). This approach was taken as the most important information to understand from the net torque is which direction will the child be moved in. Thus, this assumption about how the torque is denoted is expected to help the device simulation code run faster.

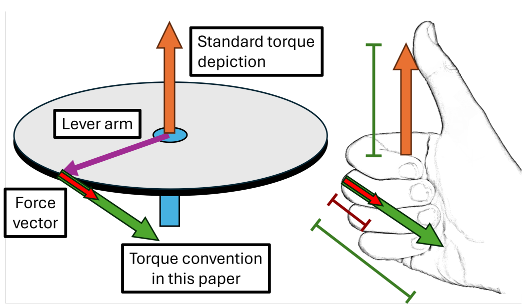


Figure 5: This figure shows how the standard right-hand rule torque vector depiction (orange arrow) compares to the convention of the torque vector used in this paper (green arrow) [24]. The orange and green arrows have the same magnitude however the torque vector convention used in this paper follows the same direction as the force vector (red arrow) acting on the lever arm. All torques discussed in this paper follow the convention depicted by the green arrow.

2.3 Genetic Optimization

The goal of the genetic optimization is to determine the least stringent mechanical properties required to make the device safe and predictable. The three mechanical properties to be determined are the rope stiffness, pretensioning, and upper pulley/mount offset. The optimization followed the process seen in Figure 6 is conducted separately for the four child and rope configuration scenarios.

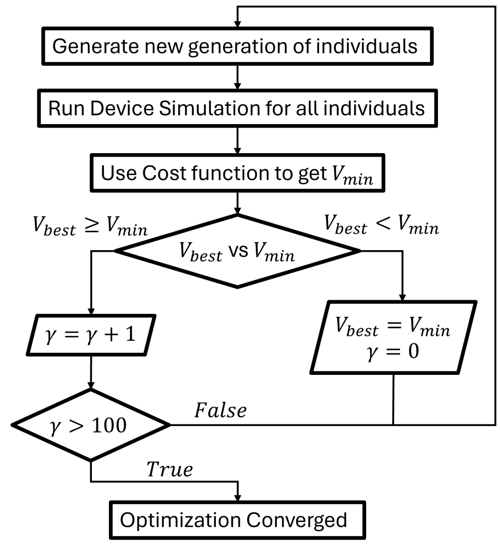


Figure 6: Cycle of the genetic optimization. V_{best} is the lowest cost for an individual found up until that point. V_{min} is the generations lowest cost individual. γ is a count of the number of generations passed in which a new lowest individual has not been found.

First, a generation of 100 individuals is created with each individual assigned values for rope stiffness, pretensioning, and upper pulley/mount offset. These mechanical properties are used to simulate the given child and rope configuration at five positions of interest (highlighted by red dots in Figure 8a). These positions of interest were selected as key points to analyse the performance of the device. The three points where the child leans left, backward, or right 22.5° are used to check if the device is safe as they represent locations where the net torque vector, on the boundary of the workspace, is weakest. The device is safe when the child cannot overpower the device with their opposition strength at any of these positions of interest. The positions where the child is leaning forward at 40°, left at 22.5°, and twisting their torso clockwise or counterclockwise 45° are used to gauge if the device is predictable. These positions represent the furthest distance the child can be from an upright seating position hence they are the location most likely to have the largest deviation. If the net torque vector at these points does not point toward the neutral position then the device is not considered predictable.

With the results of the simulation for each position of interest saved, key parameters are run through the cost function (discussed in section 2.3.1) to calculate that individual's cost. The individual with the lowest cost for that generation (V_{min}) is compared to the lowest cost individual found by the code so far (V_{best}). If V_{min} is greater than V_{best}, then the count for the number of generations passed without a new lowest individual (γ)

is incremented. Before the cycle continues, if γ is found to be greater than 100 then the optimization is considered to have converged as 100 generations have passed without a new lowest individual. The mechanical values that resulted in V_{best} are then used as the mechanical requirements for the child and rope configuration scenario being optimized for. If γ is not greater than 100, then the optimization cycle continues until this threshold is met. If V_{min} is less than V_{best}, then V_{best} is set equal to V_{min}, γ is set to zero, and the optimization cycle continues.

2.3.1 Cost function steps

The cost function is used by the genetic optimization to check if an individual results in a safe and predictable device and assigns them a cost. This is done using a linear sequence of logic checks and a cost equation (see Figure 7).

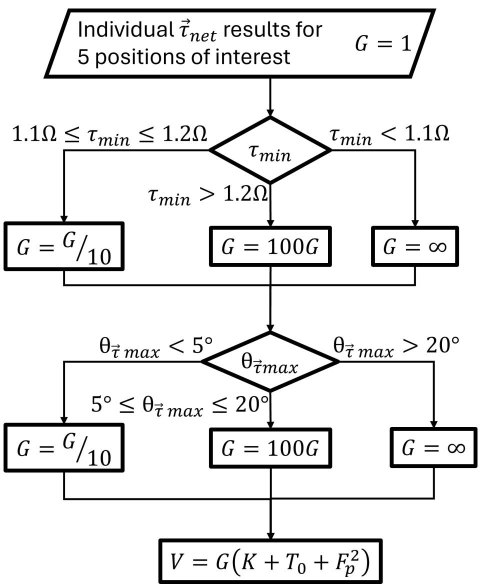


Figure 7: Logic sequence and cost equation that make up cost function. G is the global multiplier, τ_{min} is net torque with the lowest magnitude, Ω is the child's opposition strength, and $\theta_{\vec{\tau}max}$ is the angle of deviation between the net torque vector and a direct path to an upright seating position.

The logical checks compare the net torque vectors at the five points of interest to see if they meet the safety and predictability criteria. To influence the cost of an individual (V), each logic check will alter the global multiplier variable (G). This variable is used to promote or demote individuals depending on whether they meet safety and predictability criteria or not. If G is below one then the individual will have met both safety and predictability criteria. Otherwise, one or more of safety and predictability criteria were not met. The global multiplier is initially set to one so as to start with a neutral value.

The device is considered safe when the child cannot overpower the device. To quantify this, the location of interest with the net torque of the lowest magnitude (τ_{min}) is compared to the child's opposition strength (Ω). If τ_{min} is less than 110% of the child's opposition strength, then using the individual's mechanical properties makes the device too weak and unsafe hence G is set to infinity. If the net torque is greater than 120% of Ω, then the system is considered to be too strong. The global multiplier, however, is multiplied by 100. This result, though not ideal, would highlight that the optimization could not achieve the safety criteria if this individual turned out to be the best option. Finally, if the net torque lies between 110% and 120% of the child's opposition strength then the system is considered to be safe but still well tuned to the child's strength. This individual is given an incentive with G being divided by ten.

Next, the predictability of the device is evaluated. For this, the position of interest which results in a net torque vector ($\vec{\tau}_{net}$) that deviates the most from a straight line that runs from the child's current position to an upright position ($\vec{D}_t$) is checked. $\vec{D}_t$ is perpendicular to the child's torso. This angle of deviation ($\theta_{\vec{\tau}max}$) is calculated using Equation 10.

$$\theta_{\vec{\tau}max} = \arccos \frac{\vec{\tau}_{net} \cdot \vec{D}_t}{\|\vec{\tau}_{net}\| \cdot \|\vec{D}_t\|} \tag{10}$$

Once this is determined, there are three possible conditions θ may satisfy. If this value is lower than 5° then this is considered to be pointing in a predictable direction and G is divided by ten. If the angle is greater than 20° then this is considered to be poorly directed and unsafe. G is thus set to infinity. If θ lies between these two angles, then the individual solution is considered not ideal. However, is still kept a possible solution in the case no individual is able to satisfy the ideal criteria. G is multiplied by 100 to reflect this.

The resulting global multiplier is then fed into the cost equation. This will result in the cost of the individual either being set to infinity, preventing the individual from being a candidate, promoting the individual, or demoting them whilst still keeping them as a possibility. The cost equation is made of four components: the global multiplier, the individual's rope stiffness (K) and pretension (T_0) values, and the maximum trunk-parallel force calculated (F_p). The individual's rope stiffness and pre-tension values are generated by the optimization script. The maximum trunk-parallel force is included as this should also be as low as possible to make it easier to design a harness for the device. This value is squared to increase the effect of larger trunk-parallel force values as the upper limit of what this number could be is unknown.

A genetic optimization was used over other optimization techniques as it offers a technique that is more efficient than a grid search yet is not too susceptible to local minima [25]. Though it is possible to develop an individual function to describe the entire device and thus use a gradient search optimization technique, it would require a substantial rewrite of the code. As some of the code was still being developed and tweaked, a genetic optimisation was seen as an accessible optimization implementation for this paper.

2.3.2 Optimization Verification

To conduct the safety and predictability analysis, the genetic optimization for the four scenarios do not fall within a local minima. To ascertain this, the optimization for each scenario was run through the optimization cycle ten times. After this, the result with the lowest cost was selected as the best solution for that scenario. The ten results were also used to analyse if the output variables were consistent by inspecting means, medians, and standard deviations of values for the cost, rope stiffness, pre-tension, upper pulley/mount offset, lowest net torque magnitude and worst directed net torque vector of any positions of interest, and the maximum trunk-parallel force.

2.4 Analysing the Four Scenarios

Once an optimization is conducted for each of the four scenarios (three-year-old and eight-year-old child in both the interconnected and separate ropes configuration), their results are verified with force field plots. Each force field plot corresponds to the amount the child's torso is twisted. Within each plot, the green arrows represent the net torque vector ($\vec{\tau}_{net}$) acting on the child at that posture. To accurately depict the three-dimensional net torque vectors on a two-dimensional plot, the vector must be transformed. This is done using equation 11.

$$\vec{\tau}_{net,2D} = ||\vec{\tau}_{net}|| \frac{\vec{\tau}_{net}(\tau_x, \tau_y)}{||\vec{\tau}_{net}(\tau_x, \tau_y)||} \tag{11}$$

The unit vector of only the x and y components of the net torque vector ($\vec{\tau}_{net}(\tau_x, \tau_y)$) are multiplied by the magnitude of $\vec{\tau}_{net}$ to get a new, two-dimensional vector ($\vec{\tau}_{net,2D}$). $\vec{\tau}_{net,2D}$ is ultimately what is plotted in the force field plots. These plots are visually analysed to confirm that all net torque vectors point towards the neutral seating position, highlighted by a blue dot. If so, then the result is verified as safe and predictable.

Additionally, sensitivity analysis was conducted on each of the four scenarios to see how they would react when poorly configured. This was done by modifying rope stiffness and pre-tension from those produced by the optimization[2]. Both these mechanical properties were varied by ±20% to produce eight combinations of rope stiffness and pre-tension settings. A force field plot for each mechanical property combination was created to gauge how well, qualitatively, they would conform to the safety and predictability criteria. Only a 0° twist angle was used as adding more force field plots would make displaying the data visually confusing and, whilst twist angles do have some impact on the force field plot, a 0° twist angle provides a good idea of the overall system response. It is hypothesised that these other combinations may not adhere to the safety criteria but should still adhere to the predictability criteria.

[2]Rope stiffness or pre-tensioning modulation were the focus of this section as upper pulley/mount offset did not have a large impact on device performance for values above 0.3 m. Hence, to ensure that the analysis is clear, modulating the upper pulley/mount offset was not included.

2.5 Determining Mechanical Requirements

Once the four scenarios were found to be safe and predictable and not highly sensitive to changes in rope properties, then material requirements can be identified and products that satisfy these requirements can be sourced. The two most important requirements are the rope stiffness and maximum tension values. Other mechanical requirements investigated can be found in Appendix D. Furthermore, trunk-parallel forces acting on the harness are also valuable to analyse as they would have a significant impact of the future design of the harness.

2.5.1 Comparing the Two Rope Configurations

Once all other analysis is completed, the two rope configurations will be compared to try and determine which configuration is best suited to the use case of the device. The first comparison will compare if the rope configurations, for both children, successfully passed the safety and predictability criteria. The better rope configuration is able to pass both of the criteria with both children. If both rope configurations pass all criteria for both children, then what differences remain will be qualitatively analysed.

Next, comparisons between the material requirements will also be conducted to understand which configuration requires the lowest mechanical properties. If one configuration has higher mechanical requirements then that configuration will be considered worse. This is especially true in the case where off-the-shelf components cannot be found for one configuration but can be found for another.

3 Results

Table 3 shows the results of the optimization for each of the four scenarios (three-year-old and eight-year-old child in the interconnected and separate ropes configurations). The value of the best result, means, and medians were all similar numbers and the standard deviations were small with no outliers found. These results indicate that the outputs across the ten optimization cycles were consistent and indicates a global minima has been found. Additionally, all optimization cycles were able to converge on results that satisfied the safety and predictability criteria (no net torque vector on the edges of the work space has a magnitude of less than 110% of the child's opposition strength and all net torque vectors are never more than 5° off a direct path to an upright seating position).

3.1 Safe and Predictable Device Response

Looking at the force field plots (Figure 8), it was clear that each child and rope configuration condition was safe

	Variables	Three Interconnected Ropes								Three Separate ropes								Units
		Three-year-old				Eight-year-old				Three-year-old				Eight-year-old				
		Value	Mean	Median	SD	Value	Mean	Median	SD	Value	Mean	Median	SD	Value	Mean	Median	SD	
Individuals	Cost	235.9	236.9	237.0	0.6	1913.8	1919.7	1918.3	4.9	63.5	63.6	63.5	0.1	805.9	811.8	813.5	4.1	-
Individuals	Rope Stiffness	753.3	748.4	748.0	7.9	1219.8	1221.1	1223.8	15.9	745.3	747.7	746.8	3.6	1693.9	1694.8	1695.1	5.8	N/m
Individuals	Pre-Tension	45.0	45.6	45.5	0.8	110.0	109.8	109.3	3.5	107.4	107.1	107.2	0.6	158.8	158.2	158.3	1.6	N
Individuals	Upper Pulley/Mount Offset	0.478	0.477	0.478	0.003	0.413	0.414	0.414	0.005	0.700	0.700	0.700	0.000	0.459	0.455	0.455	0.003	m
Simulation Results	Maximum Tension	220.4	219.8	219.7	1.1	566.2	566.6	567.1	2.8	251.1	251.2	251.2	0.2	675.4	675.1	674.9	1.4	N
Simulation Results	Minimum Workspace Boundary Torque Magnitude	24.9	24.9	24.9	0.03	92.5	92.6	92.6	0.1	24.9	24.9	24.9	0.03	92.9	92.9	92.8	0.2	Nm
Simulation Results	Worst Torque Direction	2.329	2.316	2.316	0.017	2.355	2.355	2.363	0.032	4.088	4.102	4.095	0.024	4.999	4.969	4.977	0.032	°
Simulation Results	Maximum Trunk-parallel Force	151.0	151.3	151.3	0.2	436.0	436.6	436.5	0.6	74.1	74.2	74.2	0.1	280.6	281.6	281.9	0.7	N

Table 3: Results of the genetic optimization for each of the four scenarios. The "Value" column shows the optimization result, of the ten optimization attempts, with the lowest cost function value. The columns for mean, median, and standard deviation (SD) come from calculating that variable across the ten optimization attempts conducted per scenario. Results for each individual can be found in Appendix B

and predictable across the workspace range. All green arrows (representing the net torque vectors acting on the child) pointed nearly directly towards the upright seating position (the blue dot in the figures). For the separate ropes configuration, a relatively linear increase in net torque vector strength was noticed as the child leaned further from the neutral position. However, for the interconnected rope configuration, the most extreme lean angles resulted in larger net torque magnitudes compared to the separate ropes configuration. This was especially evident in the region where the child leans forward beyond 22.5°. For both rope configurations, a twist of ± 45° caused all net torque vectors to slightly shift in direction. This was more pronounced with the interconnected rope configuration. Even so, the measured points of interest still satisfied the directional requirements for a predictable response.

3.2 Device Sensitivity Analysis

Sensitivity analysis of the device to changes in rope stiffness and pre-tensioning values showed that, for all four scenarios, the device still remained predictable though not necessarily safe. All net torque vectors pointed towards the neutral position with seemingly little variance in direction between the different plots (Figure 9). When rope stiffness and/or pre-tension were reduced below the optimal values, the magnitude of the net torque vectors also reduced however none reduced to the point were the child would have no support.

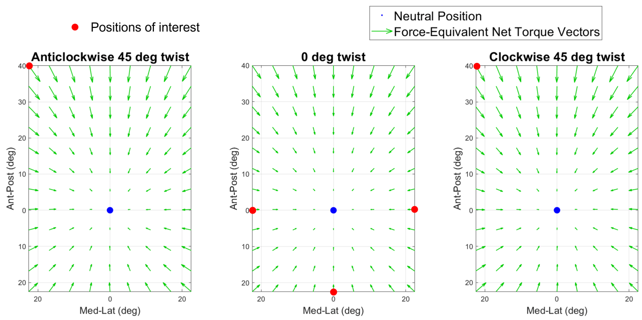
3.3 Mechanical Requirements

Maximum and minimum rope stiffness values for each rope configuration correspond to the rope stiffness values of the eight-year-old and three-year-old respectively as seen in Table 3. Maximum tension values experienced by the ropes are also present in the table when looking at the value for the eight-year-old for each rope configuration. These mechanical requirements were able to be satisfied by heavy duty shock cord supplied by Ibex Marina Ropes Ltd. [26]. For the interconnected rope configuration, ropes with diameters between 22-26 mm were required. For the separate ropes configuration, diameters between 16-26 mm were necessary (see Appendix E.1 information about the rope sourced). These ropes, however, would not be compatible with the off-the-shelf pulleys found which support ropes of a maximum diameter of 12 mm (see Appendix E.3 for more information).

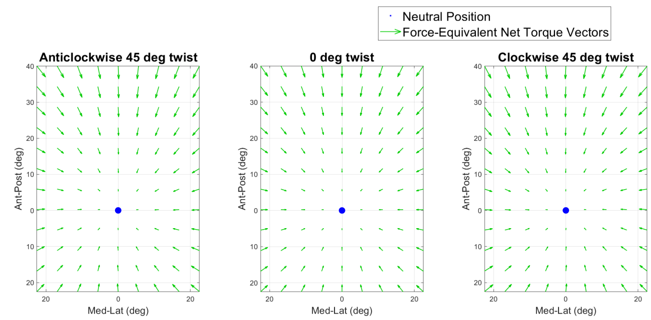
4 Discussion

The force field plots produced show some small differences between the two rope configurations for each child. The interconnected ropes design has a force field strength distribution that increases more towards the extremes resulting in a higher maximum net torque. This is may be due, in part, to the fact that this configuration has two ropes attached to each harness mounting point potentially doubling the stiffness felt on each mounting point. This may be mitigated somewhat by the rope being attached to two harness points which may relieve some of the force. It is still evident this configuration has a stronger field when looking at the far extremes of anterior lean. This could be seen as a benefit from a safety perspective as it means the device would be able to apply more force if the child was outside the workspace region thus pulling the child harder into the safe region. However, this may be a negative characteristic as it could make it too difficult for the child to lean forward as freely and/or result in a less predictable response which may make it difficult for children to get used to. To determine which response type is better, further analysis would need to be conducted to understand which configuration would suit therapists and the training programs better.

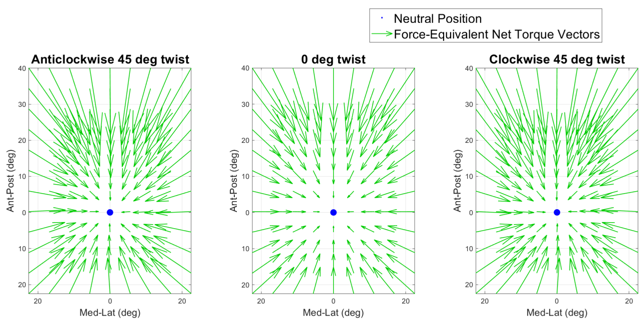
Rope stiffness and pre-tension increased between the younger and older child. This is to be expected as the older child is heavier and stronger hence the device must generate more force to compensate. However, it was no-



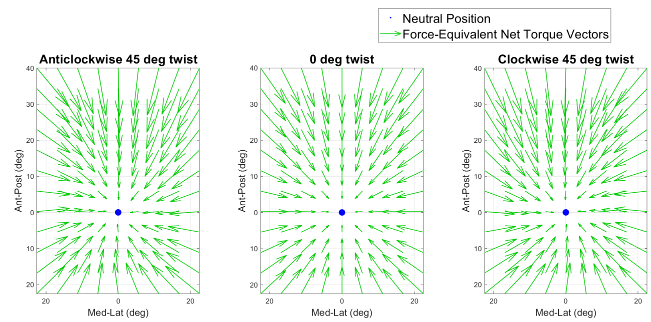
(a) Interconnected ropes configuration with three-year-old. Positions of interest are highlighted with red dots.



(b) Separate ropes configuration with three-year-old.



(c) Interconnected ropes configuration with eight-year-old.



(d) Separate ropes configuration with eight-year-old.

Figure 8: Force field plots of the four tested conditions. Positions that are not safe/predictable will show with red arrows.

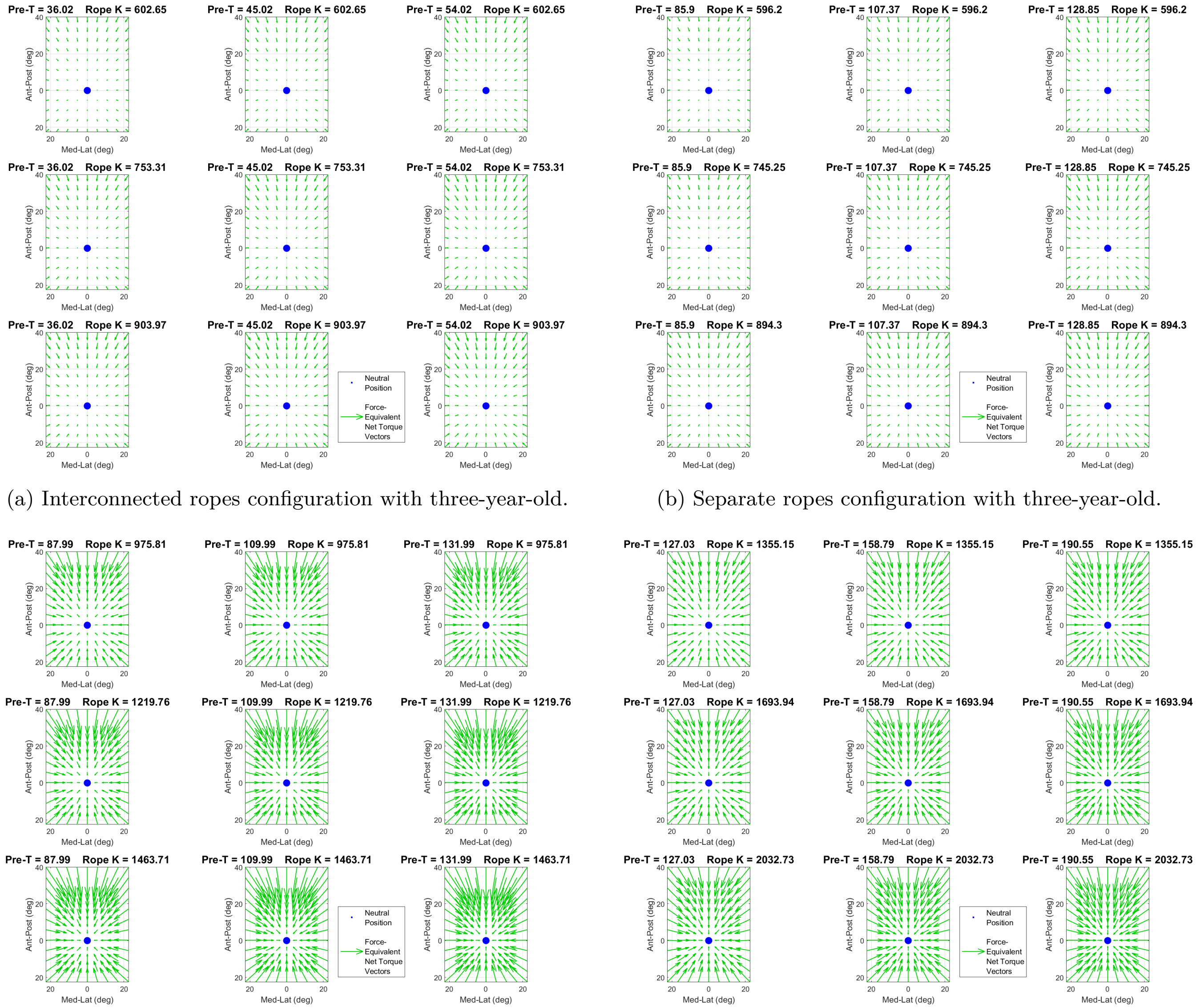
table that the upper pulley/mount offset was relatively similar between the four scenarios except for the three-year-old in the separate ropes configuration. During initial testing it was noted that higher upper pulley/mount offsets resulted in better net torques however this would result in higher trunk-parallel forces. Including trunk-parallel forces in the cost function was meant to mitigate this but, in this scenario, it seems the overall lower trunk-parallel forces did not sway the cost function. This does not pose an issue to the design however future work could investigate a device that has a more limited offset range between 0.4 to 0.5 m or even a static offset in order to simplify the design.

Both rope configurations were able to satisfy the safety and predictability requirements for both children. This indicates that the device, in theory, could provide an alternative method of supporting children whilst they train their trunk muscles. Whilst mechanical properties were able to be satisfied individually, incompatibilities were found between the rope diameters required (between 16 mm to 26 mm in diameter) and the maximum rope thickness the pulleys found could accommodate (12 mm).

One solution could be to manufacture pulleys able to accept ropes of up to 26 mm in diameter. This solution is straight forward however the thick ropes could pose other issues. Primarily, how will the stiffness of the ropes be modulated across the range of children between the ages of three and eight? In the larger pulleys solution, needing to completely replace a rope for different children could be time consuming. Especially in the case of the interconnected ropes configuration, not only would the rope need to be reconnected to the harness but it would need to be re-run through the entire pulley system. An alternative solution could be to use multiple thinner ropes which are compatible with the 12 mm maximum compatible rope diameter of the found pulleys. In this solution, the ropes would run in parallel allowing their stiffness to be summed resulting in greater overall stiffness for the combined ropes [23]. This approach has the benefit of not requiring custom components and could allow for individual ropes to be added or removed, depending on the size and weight of the child, without requiring all ropes to be disconnected. This has the potential to save therapists time during set-up if a system which allows the quick connecting and disconnecting of individual ropes could be created.

Applying this parallel ropes idea, it can be computed how many 12.5 mm ropes would be needed to achieve the maximum stiffness for both rope configurations based on the manufacturer's rope specifications (see Appendix E.1). This results in five ropes needed for the interconnected ropes configurations and four ropes needed for the separate ropes configuration to surpass the maximum stiffness of $1237.1\,N/m$ and $1687.0\,N/m$ respectively[3]. Whilst the number of ropes needed is manage-

[3]Fewer ropes are required for the separate ropes configuration even with the higher stiffness requirement because the elongation of the rope in the separate ropes configuration is



(a) Interconnected ropes configuration with three-year-old.

(b) Separate ropes configuration with three-year-old.

(c) Interconnected ropes configuration with eight-year-old.

(d) Separate ropes configuration with eight-year-old.

Figure 9: Force field sensitivity analysis plots of the four tested scenarios. Each plot represents 0° twist angle.

able, creating a rope system that allows ropes to easily be added or removed seems more difficult to implement in the interconnected ropes configuration as it inherently has more pulleys the ropes must run through. In contrast, the separate ropes configuration only has to connect the ropes to the harness and the support column.

Looking at pre-tensioning requirements, interconnected ropes configuration required lower pre-tension values of 45.4 to 106.3 N compared to 107.3 to 159.9 N for the separate ropes configuration. This means that the pre-tensioning system would not need to be quite as robust however, based on a pre-tensioning system seen in Appendix E.3, this likely would not be a large issue to overcome.

greater.

Finally, the trunk-parallel forces present in the separate ropes configuration were far lower than those of the interconnected ropes. However, trunk-parallel forces in the separate ropes configuration are still quite high which will have some consequences on the design of the harness (discussed in section 4.2).

On the whole, the separate ropes configuration appears to be the better solution mainly due to its design being simpler to modify to allow for multiple ropes. In addition, the overall lower trunk-parallel forces present in this configuration would make it easier to design a harness for the device. Finally, this configuration also has the benefit of having fewer components overall which could make it cheaper to produce and easier to maintain.

4.1 Limitations

Two key limitations of the simulation are the model of the child's body and lack of kinematic modeling. Replacing the inverse pendulum moving about a fixed point for a model of the pelvis and spine would give greater accuracy to how the child would move whilst in the device. In addition, altering the point mass to a distributed weight would further enhance the accuracy of the modeled child. Finally, being able to include the child's arms in the model would allow for further safety testing of the device.

Including kinematic modeling of the child and device would open up more analysis possibilities. Seeing the child's motion over time could better inform how the device would respond. In addition, a kinematic model would also allow the incorporation of the child's own movements. This could then be used to simulate a training exercise to better understand the loading on the harness and the support the child would get. These potential additions to the simulation would help to improve the realism and accuracy of the simulation.

4.2 Future Work

This paper was the first to analyse this theorised tendon based trunk support device and has highlighted key areas that require further research in order to understand if the device could satisfy its expected position in the market. The prominent issue is how to design the harness to allow for up to 208.6 N of trunk-parallel force as well as allow the therapist to connect the ropes to the child at different heights along the torso. Given the force is equivalent to 60% of the eight-year-old child's body weight, it is likely that a full body harness would be necessary. There are, however, a few ways in which this value could be reduced to ease the harness design. One could be to re-evaluate the size of the workspace. It was already expected that the workspace is an overestimate of how far a child with a movement disorder could lean. Therefore, finding more precise workspace lean and twist boundaries would reduce the resulting trunk-parallel forces.

The design of the ropes themselves could also be altered. Instead of relying purely on the tension in the ropes to prevent the child from falling, a rope with two different stiffnesses could be used. The low stiffness component would allow the child to move freely within the work space and the other high stiffness part would prevent the child from moving after a certain point. This stiffer component would be relied upon to prevent the child from falling. This may allow the low stiffness component to be of a lower stiffness than what was found in this paper which could drastically reduce trunk-parallel forces. Assuming such a rope design were used, research would also need to be conducted to determine how strong to make the response of the system. If it turns out the ropes still need to be very stiff then using this multi-stiffness rope may not yield much relief for the trunk-parallel force.

Another area to investigate further are other possible rope configurations. One option is to investigate the use of different rope stiffnesses in each of the ropes. This may allow the asymmetric workspace to allow for easier range of motion across the anterior-posterior axis. Different rope stiffnesses could also be used to provide more support to children who cannot lean in each direction to the same degree. Beyond rope stiffness, other possible rope configurations could be conceived that help to, for example, reduce trunk-parallel forces. This paper focused on analysing two basic configurations to gauge the overall validity of the concept but alternative configurations could, for example, involve ropes extending to the child from multiple heights from each support column. The code used in this paper could be modified to test these different options in the future.

Finally, another area to consider is the usability of the device. Questions such as how can the therapist set the pre-tension in the ropes or alter the stiffness of the ropes quickly require careful design in order for the device to fit well within the market. Additionally, how will the therapist know what settings to use for each child? This could be solved by refining the code used in this paper to allow therapists to input information and quickly get a response of recommended settings for them to use.

5 Conclusions

There is evidence to support the prospect that this novel trunk support device does show some promise. The optimized versions of the two rope configurations showed that, theoretically, the device can provide safe and predictable support to virtually any child between the ages of three and eight. The device also exhibited resilience to changes in rope stiffness and pre-tensioning values where net torque responses across the workspace still showed the device would support the child. Finally, the calculated mechanical requirements seemed to be attainable however a hurdle was posed by the sourced ropes having too big a diameter for the found pulleys. This could be rectified either by creating customised components that can accommodate the width of the thick ropes or by using multiple ropes. Based on the information found within this paper, following the second option would likely be easier to implement especially for the three separate ropes configuration. It was concluded that this rope configuration was the more suitable configuration between the two options. As this is the first paper to investigate such a device, there are still a myriad of topics that require further research. The most pressing is a harness design that could comfortably withstand the high trunk-parallel forces.

References

[1] I. Krägeloh-Mann and C. Cans, "Cerebral palsy update," Brain and Development, vol. 31, no. 7, pp. 537–544, 8 2009.

[2] E. N. Marieb and K. Hoehn, Human Anatomy & Physiology, 10th ed. Pearson Education Limited, 2016.

[3] R. Dewar, S. Love, and L. M. Johnston, "Exercise interventions improve postural control in children with cerebral palsy: a systematic review," Developmental Medicine & Child Neurology, vol. 57, no. 6, pp. 504–520, 6 2015. [Online]. Available: https://onlinelibrary.wiley.com/doi/full/10.1111/dmcn.12660

[4] R. Sæther, J. L. Helbostad, L. Adde, S. Brændvik, S. Lydersen, and T. Vik, "The relationship between trunk control in sitting and during gait in children and adolescents with cerebral palsy," Developmental Medicine & Child Neurology, vol. 57, no. 4, pp. 344–350, 4 2015. [Online]. Available: https://onlinelibrary.wiley.com/doi/full/10.1111/dmcn.12628

[5] WHO Multicentre Growth Reference Study Group, "Windows of achievement for six gross motor milestones," 2006. [Online]. Available: https://www.who.int/tools/child-growth-standards/standards/motor-development-milestones

[6] D. J. Curtis, P. Butler, S. Saavedra, J. Bencke, T. Kallemose, S. Sonne-Holm, and M. Woollacott, "The central role of trunk control in the gross motor function of children with cerebral palsy: a retrospective cross-sectional study," Developmental Medicine & Child Neurology, vol. 57, no. 4, pp. 351–357, 4 2015. [Online]. Available: https://onlinelibrary.wiley.com/doi/full/10.1111/dmcn.12641

[7] E. I. Rodríguez-Grande, O. C. Vargas-Pinilla, M. R. Torres-Narvaez, and N. Rodríguez-Malagón, "Neuromuscular exercise in children with Down Syndrome: a systematic review," Scientific reports, vol. 12, no. 1, 12 2022. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/36056081/

[8] M. A. Elshafey, M. S. Abdrabo, and R. K. Elnaggar, "Effects of a core stability exercise program on balance and coordination in children with cerebellar ataxic cerebral palsy," Journal of Musculoskeletal & Neuronal Interactions, vol. 22, no. 2, p. 172, 6 2022. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9186458/

[9] Y. T. Cheng, T. K. Wong, W. W. Tsang, C. M. Schooling, S. S. Fong, D. Y. Fong, Y. Gao, and J. W. Chung, "Neuromuscular training for children with developmental coordination disorder: A randomized controlled trial," Medicine, vol. 98, no. 45, p. e17946, 11 2019. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6855479/

[10] N. Belizón-Bravo, R. P. Romero-Galisteo, F. Cano-Bravo, G. Gonzalez-Medina, E. Pinero-Pinto, and C. Luque-Moreno, "Effects of Dynamic Suit Orthoses on the Spatio-Temporal Gait Parameters in Children with Cerebral Palsy: A Systematic Review," Children, vol. 8, no. 11, 11 2021. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8621824/

[11] A. F. Bailes, K. Greve, and L. C. Schmitt, "Changes in two children with cerebral palsy after intensive suit therapy: A case report," Pediatric Physical Therapy, vol. 22, no. 1, pp. 76–85, 3 2010. [Online]. Available: https://journals.lww.com/pedpt/Fulltext/2010/02210/Changes_in_Two_Children_with_Cerebral_Palsy_After.11.aspx

[12] E. Martins, R. Cordovil, R. Oliveira, J. Pinho, A. Diniz, and J. R. Vaz, "The immediate effects of a dynamic orthosis on gait patterns in children with unilateral spastic cerebral palsy: A kinematic analysis," Frontiers in Pediatrics, vol. 7, no. FEB, p. 42, 2019. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6393373/

[13] K. Kaushik and K. Kumar, "Effect of Cage Therapy using Advanced Spider Suit Compared to Traditional Physical Therapy on Gross Motor Function in Children with Cerebral Palsy – An Indian Experience," International Journal of Neurorehabilitation, vol. 03, no. 01, 2016.

[14] "FORTIS-102." [Online]. Available: http://fortiseng.fortis.or.kr/goods/view/CODTS-0450

[15] P. Herrero, E. M. Gomez, M. A. Asensio, E. García, R. Casas, E. Monserrat, and A. Pandyan, "Study of the therapeutic effects of a hippotherapy simulator in children with cerebral palsy: A stratified single-blind randomized controlled trial," Clinical Rehabilitation, vol. 26, no. 12, pp. 1105–1113, 5 2012. [Online]. Available: https://journals.sagepub.com/doi/10.1177/0269215512444633?url_ver=Z39.88-2003&rfr_id=ori%3Arid%3Acrossref.org&rfr_dat=cr_pub++0pubmed

[16] J. H. Park and J. H. You, "Innovative robotic hippotherapy improves postural muscle size and postural stability during the quiet stance and gait initiation in a child with cerebral palsy: A single case study," NeuroRehabilitation, vol. 42, no. 2, pp. 247–253, 1 2018.

[17] V. Santamaria, T. Luna, M. Khan, and S. Agrawal, "The robotic Trunk-Support-Trainer (TruST) to measure and increase postural workspace during sitting in people with spinal cord injury," Spinal Cord Series and Cases 2020 6:1, vol. 6, no. 1, pp. 1–7, 1 2020. [Online]. Available: https://www.nature.com/articles/s41394-019-0245-1

[18] V. Santamaria, X. Ai, and S. K. Agrawal, "A motor learning-based postural intervention with a robotic trunk support trainer to improve functional sitting in spinal cord injury: case report," Spinal Cord Series and Cases 2022 8:1, vol. 8, no. 1, pp. 1–9, 11 2022. [Online]. Available: https://www.nature.com/articles/s41394-022-00554-2

[19] R. J. Palisano, L. Avery, J. W. Gorter, B. Galuppi, and S. W. McCoy, "Stability of the Gross Motor Function Classification System, Manual Ability Classification System, and Communication Function Classification System," Developmental Medicine and Child Neurology, vol. 60, no. 10, pp. 1026–1032, 10 2018. [Online]. Available: https://onlinelibrary.wiley.com/doi/full/10.1111/dmcn.13903https://onlinelibrary.wiley.com/doi/abs/10.1111/dmcn.13903https://onlinelibrary.wiley.com/doi/10.1111/dmcn.13903

[20] J. Molenbroek and T. Huysmans, "DINED Anthropometric Database," 1993. [Online]. Available: https://dined.io.tudelft.nl/en/database/tool

[21] R. K. Jensen, "Body segment mass, radius and radius of gyration proportions of children," Journal of Biomechanics, vol. 19, no. 5, 1986.

[22] R. M. Alexander, " Biomechanics and Motor Control of Human Movement. David A. Winter ," The Quarterly Review of Biology, vol. 66, no. 1, 1991.

[23] H. Young and R. Freedman, University physics with modern physics, 2007.

[24] R. C. Hibbeler, Mechanics of Materials Eighth Edition, 8th ed. Pearson, 2001.

[25] S. S. Rao, Engineering optimization: Theory and practice, 4th ed., 2009.

[26] Ibex Marina Ropes Ltd, "BS3F70 Heavy Duty Shock Cord." [Online]. Available: https://www.ibexmarina.com/portfolio-item/heavy-duty-shock-cord/

[27] Bosch Rexroth, "Bosch Rexroth Silver Aluminium Profile Strut, 20 x 20 mm, 6mm Groove, 3000mm Length." [Online]. Available: https://uk.rs-online.com/web/p/tubing-and-profile-struts/4667219

[28] FOCCAR, "Single Pulley Block | Silent Gym Cable Pulley - Rotation 360 Degree, Detachable, Smooth, Durable, DIY Attachment for DIY Gym Pulley System, Equipment Replacement, Hoists. Lear-au." [Online]. Available: https://www.amazon.co.uk/Single-Pulley-Block-Silent-Cable/dp/B0C7V6M8QR?th=1

[29] CMC and Harken Industrial, "CMC CLUTCH™ BY HARKEN INDUSTRIAL™." [Online]. Available: https://www.cmcpro.com/equipment/cmc-clutch-by-harken-industrial/#learn_more

[30] SELEWARE Store, "SELEWARE 1000 lb Capacity Wall Ceiling Mount Bracket for Suspension Strap Trainer, Suspension Kit for Yoga Swing, Hanging Hardware for Battle Ropes, Woss Suspension, Olympic Rings, Gym Accessories." [Online]. Available: https://www.amazon.com/SELEWARE-Capacity-Suspension-Hardware-Accessories/dp/B07S3T6GH9/ref=sr_1_36?dib=eyJ2IjoiMSJ9.wf8pkwRke8zDwGKCa_QttYLHfndZrEzhT9_7GuWXtseKnjR21xy8SDldTIDjrU-1rTBDdYZaL5CnCRV1Zv_ECQi1bs5_PgFDKgxB_hWa0W4Q11ZuE9VvOfR1qKpSBvMNOlxfIyLUgHgCHAmq-2drRQMkljVnfD_8GLnxwdS7Zo2xO-J1oZfC7gRmvsI_7sJ8AIS2g4APkraJTrnm-BP_bh5Ie3SV5X7J5wrMD2h5dmQ1V8ty3omzjJJKmMwCwXkYhF72HxE1ic_HH0uISOvzavM0_CGkbKDZHclT_a01XV4.EeIQ--eQRoF7MlQ1VNw_WY_UOJvmXCcgURKJzgWrs2Y&dib_tag=se&keywords=battle%2Brope%2Bwall%2Bmount&qid=1711033625&sr=8-36&th=1

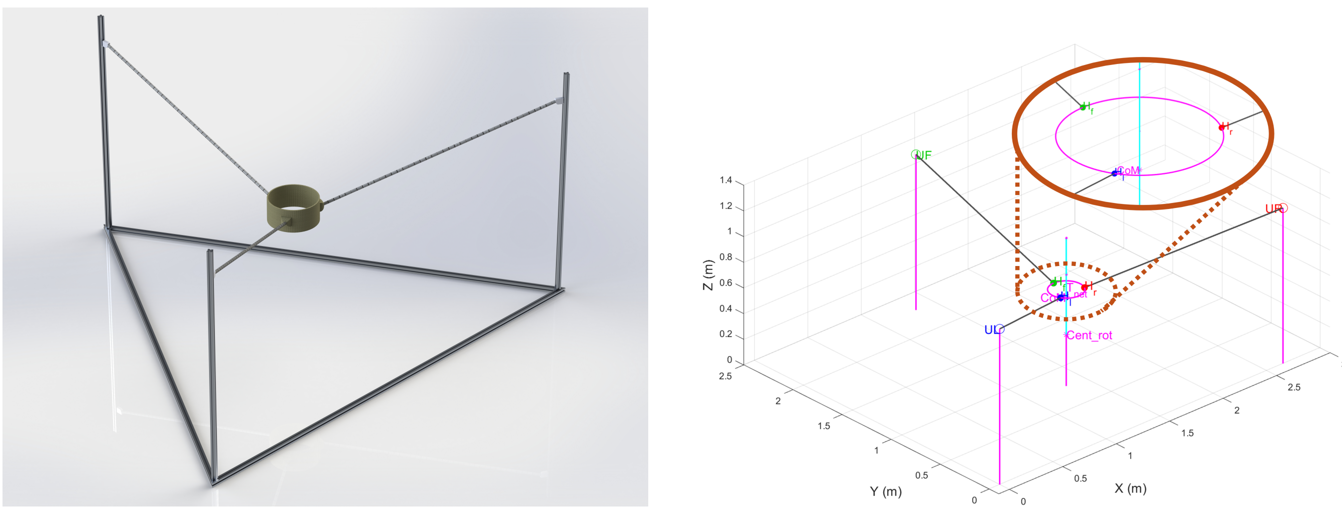
Appendix

A Simulation Steps In-Depth Explanation

Step 1: Define location of components

The location, in 3D space, of all points of interest must first be defined. Inputs such as known geometric information and patient anthropometry measures are used to define all points of interest. First, locations of the columns in the x-y plane are determined. This is based off the safe minimum radius, described in section 1.4, from a central position and assumes the three columns are evenly spaced with one column placed directly in front of the child. With the x-y location of each column known, the location of the pulleys can then be determined. The rope configuration used will determine the number and location of the pulleys required per column. If an interconnected rope system is being tested, then four pulleys are created per column. Two pulleys are located at the bottom of the columns and two at a height determined by the height of the harness (z coordinate) and the upper pulley/mount offset (see Figure 1a). If a separated rope system is being tested, then only one mounting point is created per column (see Figure 1b).

After the pulleys are defined, four harness points are determined. The first is the central harness point located at the desired harness height and the center of the torso. This acts as the center of the harness around which three harness points are created. These points are where the ropes would be attached to the harness. As the harness is assumed to be horizontal and circular, these three points are evenly spaced similar to the support columns. The distance these harness points radiate out from the central harness point is determined by the radius of the patient's torso. Additionally, points for the child's center of rotation and center of mass are set along a vertical line that represents the child's torso. This line intersects the central harness point. The center of rotation lies on top of the chair and the center of mass is located 62.6% along the torso (see section 2.1 for more information).



(a) CAD model

(b) 3D simulation model

Figure 10: Comparison between CAD model and the simulation model of the three separate ropes configuration.

Step 2: Determine unstretched rope length

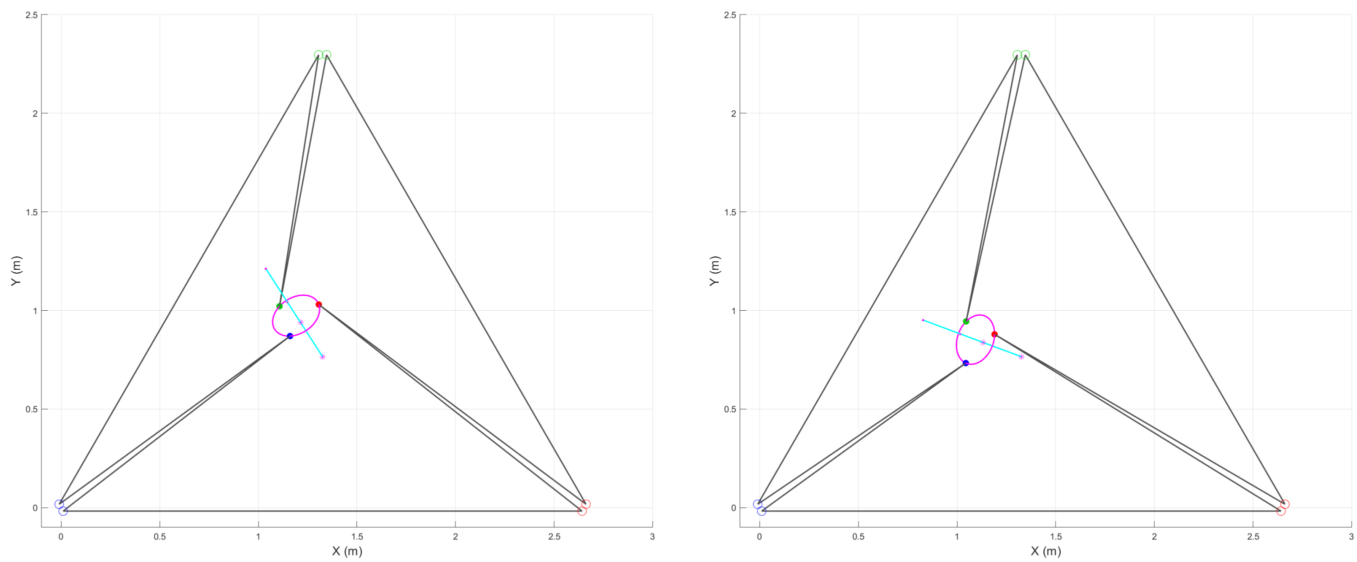
With the location of the pulleys and the starting location of the harness known, the stretched length of each rope (L_0) can be calculated by summing the distances between the harness points and pulleys/mount each rope runs through/to. Taking the computed stretched length and the given inputs of rope stiffness (K) and pre-tensioning (T_0), Hooke's law can be used to determine the amount the rope is stretched (l_Δ) (see Equation 14). After this, the rope's unstretched length (L_u) can be found with Equation 13. The unstretched rope length is used to specify the minimum length of the ropes required for the device as well as used to ensure the stretch of the rope does not exceed the maximum stretch specified by the rope manufacturer. If the rope's starting length stretches beyond its maximum elongation specification, then this is flagged but the simulation continues.

$$l_\Delta = \frac{T_0}{K} \quad (12) \qquad\qquad L_u = L_0 - l_\Delta \quad (13)$$

Step 3: Define new patient coordinates based on lean and twist input

Using the given anterior-posterior lean (ϕ_{ap}), medio-lateral lean (ϕ_{ml}), and twist inputs (θ), new locations for the center of mass and four harness points can be determined using a rotation matrix (see Equation 14). XYZ order was used as the more conventional ZYX order seemed to produce unusual results (see Figure 11). Changing the origin of each point of interest to the center of rotation, each point ($\vec{P}$) can be multiplied by the rotation matrix to find their new location in 3D space ($\vec{P'}$). Once complete, each point can have their origin reverted to the global origin.

$$\vec{P'} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi_{ap} & -\sin\phi_{ap} \\ 0 & \sin\phi_{ap} & \cos\phi_{ap} \end{bmatrix} \cdot \begin{bmatrix} \cos\phi_{ml} & 0 & \sin\phi_{ml} \\ 0 & 1 & 0 \\ -\sin\phi_{ml} & 0 & \cos\phi_{ml} \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \vec{P} \quad (14)$$



(a) XYZ rotation matrix used

(b) ZYX rotation matrix used

Figure 11: Comparison between the results of the two different orders to create a rotation matrix. For this example, posture inputs are given as 40° lean forward, 22.5° lean left, and 45° twist counterclockwise. The top of these figures is the front of the device and the green dot on the harness represents the direction the child is facing after setting the posture.

Step 4: Determine new rope state

With the child's new location calculated, the tension and length of each rope must be recalculated. First, the length of the ropes are re-analysed using the same method used to find their starting stretched length (see Step 1). Next, the difference in length between each rope's new length and their unstretched length is calculated. This is then input into Hooke's law, along with the rope stiffness, to find the tension present in each rope (now T_1) (see Equation 14). During this stage, the stretch of the rope is again compared to the elongation limit specified by the rope manufacturer. If the rope is stretched beyond its specification, then this is flagged but the calculations continue.

Step 5: Determine net force acting at harness points and center of mass

With the tension in the ropes known, the forces acting on the outer three harness points can be calculated (see Equation 15). For each rope, a direction vector is calculated between the harness point it is attached to ($\vec{P_h}$) and the corresponding upper pulley/mount location ($\vec{P_s}$). The unit vector of this is also calculated. Then the tension within the rope (T_1) is multiplied to this vector to find the force vector acting on that harness point by that rope ($\vec{F_h}$). In the interconnected rope case, two ropes are acting on each harness point. In this situation, these two force vectors are summed together to obtain the net force on each harness point. The net gravitational force vector takes a unit vector directed straight downward and multiplies it by the weight of the upper body of the child times the acceleration of gravity ($9.81\,{}^{m}/{}_{s^2}$). This is taken to act at the center of mass of the child.

$$\vec{F}_h = \sum_{i=1}^{n} T_{1,i} \frac{\vec{P}_{s,i} - \vec{P}_{h,i}}{\|\vec{P}_{s,i} - \vec{P}_{h,i}\|} \tag{15}$$

Step 6: Decompose net force vectors into torque generating and trunk-parallel forces

With the net force vectors acting on each harness point known, each net force vector is decomposed into the components that generate a torque and a trunk-parallel force. The trunk-parallel force component ($\vec{F}_p$) is calculated by finding the projection of the net force vector ($\vec{F}_h$) that runs parallel to the trunk of the child ($\vec{V}_t$) (see Equation 16). With the trunk-parallel force vector known, it can be subtracted from the net force vector to give the torque generating vector component ($\vec{F}_\tau$) (see Equation 17). The same strategy is used to find the torque generating component of the net gravitational force.

$$\vec{F}_p = \frac{\vec{F}_h \cdot \vec{V}_t}{\|\vec{V}_t\|^2} \cdot \vec{V}_t \tag{16}$$

$$\vec{F}_\tau = \vec{F}_h - \vec{F}_p \tag{17}$$

Step 7: Determine net torque and other outputs

Finally, with the torque generating components acting at each harness point and center of gravity computed, the net torque acting on the child can be determined. First, the net torque generating force on the harness is found by summing the torque generating components acting at each harness point ($\vec{F}_{\tau,l}$, $\vec{F}_{\tau,r}$, $\vec{F}_{\tau,f}$ for the left, right, and front harness point respectively). This is then multiplied by the length from the center of rotation to the harness point (L_h) to find the torque they collectively generate. Then, the torque generated by gravity is found by multiplying the length between the center of mass and center of rotation ($0.626L_t$, see section 1.4.1) to the torque generating component of the gravity force ($\vec{F}_{\tau,g}$). Then, these two products can be summed to find the net torque acting on the child ($\vec{\tau}_{net}$).

$$\vec{\tau}_{net} = L_h(\vec{F}_{\tau,l} + \vec{F}_{\tau,r} + \vec{F}_{\tau,f}) + 0.626L_t \cdot \vec{F}_{\tau,g} \tag{18}$$

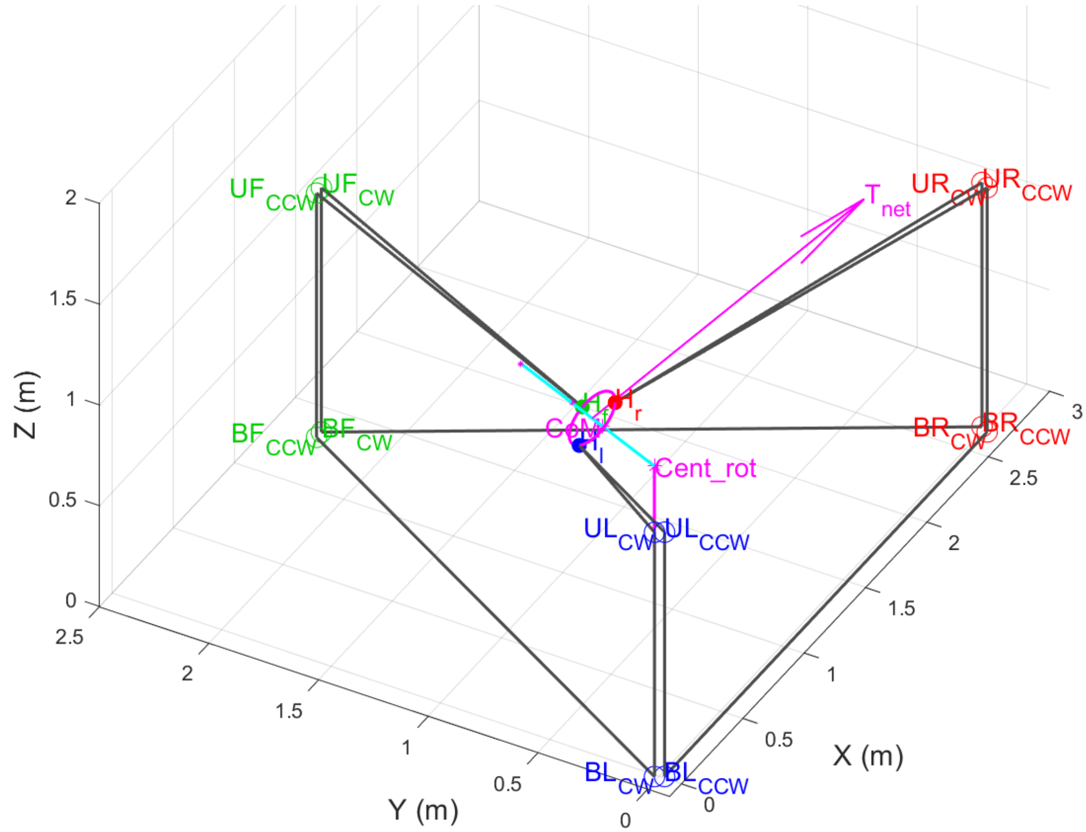


Figure 12: 3D simulation model example for three interconnected ropes configuration where child is leaning forward 40°, left 22.5°, and twisting 45° clockwise.

B Optimization results for the four tested scenarios

Variable	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Units
Cost	236.9	236.1	237.6	235.9	237.0	236.8	236.5	237.7	237.4	237.1	-
Rope Stiffness	755.6	739.3	758.5	753.3	747.4	758.1	742.1	744.7	736.3	748.6	$^{N}/_{m}$
Pre-Tension	44.9	46.5	44.9	45.0	45.8	44.7	46.0	45.7	47.0755	45.4	N
Upper Pulley/Mount Offset	0.479	0.473	0.478	0.478	0.473	0.479	0.476	0.481	0.472	0.480	m
Maximum Tension	220.8	218.5	221.5	220.4	219.7	221.2	218.8	219.1	218.3	219.7	N
Minimum Workspace Boundary Torque Magnitude	24.9	24.9	25.0	24.9	25.0	24.9	24.9	25.0	24.9	25.0	Nm
Worst Torque Direction	2.330	2.299	2.335	2.329	2.317	2.337	2.304	2.305	2.287	2.315	°
Maximum Trunk-parallel Force	151.3	151.1	151.5	151.0	151.4	151.2	151.2	151.6	151.5	151.4	N

Table 4: Ten results of the optimization cycles for the three-year-old in the interconnected ropes configuration.

Variable	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Units
Cost	1923.3	1917.6	1915.3	1927.6	1917.4	1913.8	1915.5	1919.1	1920.7	1927.0	-
Rope Stiffness	1207.1	1227.8	1216.4	1240.1	1229.1	1219.8	1232.7	1195.3	1201.5	1241.3	N/m
Pre-Tension	112.7	108.6	110.5	105.2	108.2	110.0	107.5	115.4	114.5	105.7	N
Upper Pulley/Mount Offset	0.413	0.414	0.414	0.424	0.414	0.413	0.416	0.408	0.408	0.421	m
Maximum Tension	564.1	567.9	565.5	569.6	568.0	566.2	568.7	562.2	563.6	570.5	N
Minimum Workspace Boundary Torque Magnitude	92.6	92.6	92.5	92.5	92.5	92.5	92.6	92.5	92.6	92.7	Nm
Worst Torque Direction	2.324	2.372	2.348	2.387	2.376	2.355	2.381	2.302	2.313	2.389	°
Maximum Trunk-parallel Force	437.1	436.4	436.1	437.5	436.4	436.0	436.1	436.6	436.8	437.4	N

Table 5: Ten results of the optimization cycles for the eight-year-old in the interconnected ropes configuration.

Variable	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Units
Cost	63.7	63.6	63.5	63.5	63.5	63.5	63.7	63.5	63.7	63.5	-
Rope Stiffness	754.4	749.1	745.3	743.0	745.3	747.5	753.0	748.1	746.1	745.2	N/m
Pre-Tension	106.0	106.8	107.4	107.7	107.4	107.0	106.1	106.8	107.8	107.6	N
Upper Pulley/Mount Offset	0.700	0.700	0.700	0.700	0.700	0.700	0.700	0.700	0.700	0.700	m
Maximum Tension	251.4	251.3	251.1	250.9	251.1	251.1	251.3	251.1	251.6	251.2	N
Minimum Workspace Boundary Torque Magnitude	24.9	24.9	24.9	24.9	24.9	24.9	24.9	24.9	25.0	24.9	Nm
Worst Torque Direction	4.146	4.111	4.088	4.075	4.086	4.102	4.138	4.109	4.080	4.083	°
Maximum Trunk-parallel Force	74.2	74.2	74.1	74.2	74.1	74.2	74.2	74.1	74.3	74.2	N

Table 6: Ten results of the optimization cycles for the three-year-old in the separate ropes configuration.

Variable	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Units
Cost	813.9	806.7	813.9	809.4	805.9	807.3	813.2	815.3	816.3	816.0	-
Rope Stiffness	1698.3	1692.1	1703.9	1697.0	1693.9	1690.3	1696.3	1701.7	1689.1	1685.0	$^N/_m$
Pre-Tension	156.4	157.6	155.2	159.8	158.8	158.0	157.5	158.6	160.5	159.8	N
Upper Pulley/Mount Offset	0.453	0.456	0.454	0.460	0.459	0.456	0.454	0.457	0.454	0.451	m
Maximum Tension	674.3	673.7	674.9	677.3	675.4	673.5	674.9	677.6	675.6	673.8	N
Minimum Workspace Boundary Torque Magnitude	92.7	92.6	92.8	93.2	92.9	92.6	92.8	93.3	93.0	92.7	Nm
Worst Torque Direction	4.969	4.985	4.993	4.994	4.999	4.974	4.966	4.980	4.922	4.904	°
Maximum Trunk-parallel Force	282.0	280.7	282.0	281.2	280.6	280.9	281.9	282.3	282.5	282.4	N

Table 7: Ten results of the optimization cycles for the eight-year-old in the separate ropes configuration.

C Effect of altering harness height

C.1 Introduction

With the cylindrical design of the harness, it could be placed anywhere along the torso of the child. For the range harness heights, it was assumed that it would be unlikely for a therapist to want the harness placed lower than halfway up the child's torso. The upper limit of 90% approximates the height where the arms prevent the harness from being mounted higher.

C.2 Method

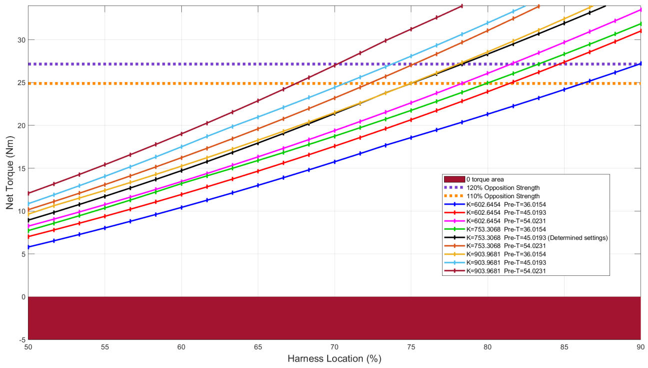
There was another sensitivity analysis was done to understand how the height of the harness affects the strength of the devices response. The harness' cylindrical design allows it to move up and down the torso of the child. In this analysis, it was assumed that the harness can be set between 50% to 90% of the child's torso height. To understand the strength of the system, the magnitude net torque acting on the child leaning backwards at 22.5° was used. In addition, each of the eight rope stiffness and pre-tension combinations used during the sensitivity analysis of each scenario were also checked. Only a magnitude of the net torque was used as it was found that the device would remain predictable thus direction would not need to be checked for this test.

C.3 Results

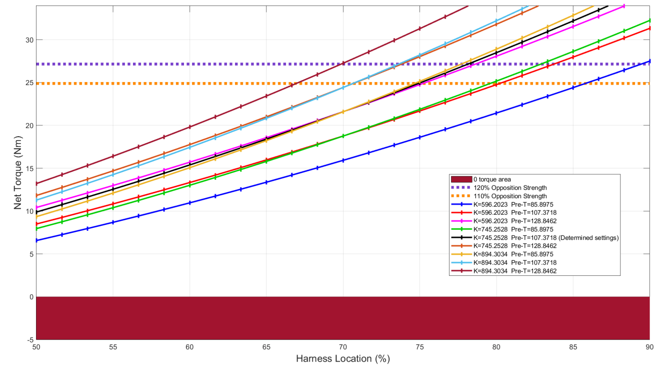
When comparing the net torque for each sensitivity condition when the child leaned backwards at 22.5° (Figure 13), there were two notable traits. The first was that, for all possible harness locations along the child's torso, the resulting net torque was able to provide some amount of force that would pull the child in the correct direction. This was indicated by all net torque values being greater than zero. Second was that each configuration had some range of harness heights which falls within the ideal zone for device strength based on each child's opposition strength (between 110% and 120%).

C.4 Discussion

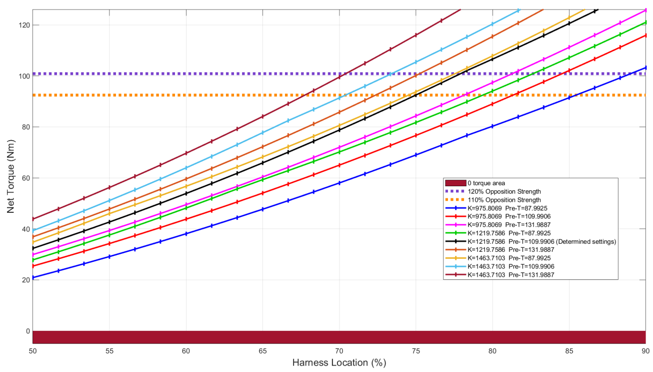
Showing that, for any harness height, the child will still have some support helps to improve trust in the device. If therapists know that, at any location they put the harness, the child will be supported they can focus more on



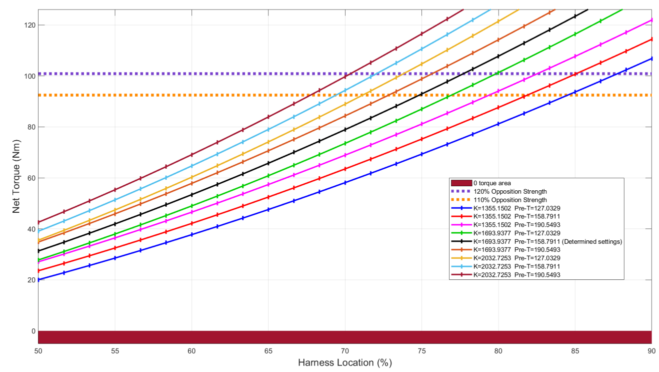
(a) Interconnected ropes configuration with three-year-old



(b) Separate ropes configuration with three-year-old



(c) Interconnected ropes configuration with eight-year-old



(d) Separate ropes configuration with eight-year-old

Figure 13: Plots of net torque acting on child when leaning backwards 22.5° across the range of harness heights for each tested condition.

providing therapy. Additionally, the flexibility of putting the harness at different heights could make the device easier to tailor to each individual child. If a child is fairly competent then the harness could be mounted lower so as to support them less forcing them to support themselves more. Alternatively, mounting the harness higher provides more robust support or would create a stronger field which a child could train by pushing against.

D Further investigation of mechanical properties

D.1 Introduction

Beyond the mechanical requirements discussed in the paper, there are five other requirements which were analysed. These include; the minimum height of the support columns, maximum tension within a rope, maximum load a pulley or mount may need to withstand, minimum length of each rope, and maximum elongation experienced by any rope.

D.2 Method

The first is minimum height of the support columns. This is calculated by adding the height of the harness, mounted 90% up the eight-year-old child's torso, above the ground to the upper pulley/mount offset and 0.1 m, which serves as a safety factor.

Another is analysing the maximum tension within any rope. For this, the optimal solution for both eight-year-old scenarios was used. Tension within each rope was analysed, for both configurations, when the child was leaning towards the positions of interest where the child leaned forward 40°, left 22.5°, and twisting 45°. Which ever rope had the highest tension is set as the maximum tension any rope must be able to cope with

Next, maximum load acting on a pulley/mount was investigated. This uses the maximum tension within any rope found above. For the interconnected ropes configuration, as there are many different directions the ropes may pull on the pulleys, it is assumed that the maximum force a pulley would need to withstand is the tension within the most loaded rope times two. This is designed to emulate a theoretical worst case where the rope goes into the pulley from one direction and exists it from the same direction. Though this is not possible during training, it serves as a buffer to ensure a pulley would be strong enough. For the separate ropes configuration, the maximum force the mount would withstand is the tension within the most loaded rope.

Finally, the minimum rope length is found by taking this result calculated during the device simulation. All four of the optimized scenarios were compared to find the minimum rope length needed per device configuration. The maximum elongation also uses this method to find its values per rope configuration

D.3 Results

Taking the optimization outputs from each condition, results for general structural requirements can be seen in Table 8. The interconnected ropes configuration showed a higher maximum pulley load, lower maximum rope tension, and longer ropes than the separate rope configuration. When looking at column bending, the interconnected rope saw greater degree of bending though this still remained under the 1 mm target. Looking at intrinsic mechanical properties (see Table 3), the interconnected rope configuration had a narrower rope stiffness range with a lower maximum stiffness than the alternative configuration. The interconnected rope configuration also saw overall lower pre-tension values and a narrower range of upper pulley/mount offsets which, additionally, saw a lower maximum and minimum offset value.

Variable	Interconnected	Separate	Unit
Minimum Column Height	1.6		m
Maximum Load on Pulley	1139.3	679.2	N
Maximum Rope Tension	569.6	679.2	N
Minimum Rope length	7.8	1.5	m
Maximum Rope Extension	6.0	29.0	%
Rope Stiffness (3YO)	746.9	745.0	N/m
Rope Stiffness (8YO)	1237.1	1687.0	N/m
Maximum Column Bending Displacement	0.929	0.554	mm

Table 8: These variables cover some key requirements for the structure of the device based on the outputs of the optimizations.

E Mechanical Components that Could Plausibly be Used in a Physical Prototype

E.1 Rope selection

A supplier of heavy duty shock cord was found. They offer a range of rope diameters that provide different strengths. Table 9 is a reproduced version of what is available on their website describing the force generated when a rope is elongated to specific elongation percentages [26]. This table includes elongation information as the rope is not linearly elastic.

Diameter (mm)	10% Minimum (N)	30% (N)		75% (N)		Total extension (%)
		Min	Max	Min	Max	
5	20	29	38	50	65	105
6.5	40	58	76	100	130	105
8	60	88	116	150	196	105
9.5	80	120	170	210	280	105
12.5	150	210	280	370	480	105
16	240	350	460	600	800	105
19	340	500	650	850	1100	105
22	460	660	880	1150	1500	105
26	640	930	1250	1600	2100	105
28	740	1080	1425	1850	2425	105
32	970	1410	1875	2425	3200	105

Table 9: The assortment of ropes supplied by the selected manufacturer [26]. Values indicate the tension within the rope when elongated to a percentage of the rope's overall length.

To determine the force generated when the rope is elongated by the percentages seen in 8, a linear extrapolation is conducted using Equation 19.

$$T = T_0 + \frac{\epsilon - \epsilon_0}{\epsilon_1 - \epsilon_0}(T_1 - T_0) \tag{19}$$

ϵ is the maximum elongation percentage calculated for the corresponding rope configuration being analysed. ϵ_0 and ϵ_1 are the elongations of 10% and 30% respectively as these two percentages are closest to the values of ϵ calculated. T_0 and T_1 are the tensions specified by the manufacturer when each rope is elongated by 10% or 30% respectively. Finally, T is the tension within the rope for the specified value of ϵ. Values for the resulting minimum generated tension, for each available rope diameter, for the two rope configurations are shown in Table 10.

With the force within each rope calculated, the stiffness of each rope diameter can be calculated. This is not a one to one comparison between stiffness's as Hooke's law is assumed within the simulation but, in reality, the stiffness of the rope is also affected by its length and how long it has been extended. Even so, looking at the stiffness of the rope under its most extreme load will give an indication if the rope likely suitable for the given application. To calculate stiffness of the ropes, Equation 20 must be used where K is the calculated stiffness, T is the tension within the rope, ϵ is the elongation percentage (the same as was used for Equation 19), and L is the length of the rope as seen in Table 8.

$$K = \frac{T}{\epsilon \cdot L} \tag{20}$$

Values for the resulting rope stiffness, for each available rope diameter, for the two rope configurations are shown in Table 10.

Diameter (mm)	Interconnected		Separate	
	Tension (N)	Stiffness ($^N/_m$)	Tension (N)	Stiffness ($^N/_m$)
5	18	39	29	67
6.5	36	77	57	133
8	54	116	87	202
9.5	72	153	118	275
12.5	138	293	207	483
16	218	464	344	804
19	308	655	492	1148
22	420	893	650	1517
26	582	1238	915	2137
28	673	1429	1063	2481
32	883	1876	1388	3240

Table 10: Tension and stiffness's of each available rope when applying the calculated rope elongation and length for each rope configuration

E.2 Elastic ropes in parallel

As only very thick ropes could, on their own, supply the necessary stiffness, it is suggested that ropes can be coupled in parallel in order to achieve the necessary stiffness. Stiffness's of ropes in parallel can be expressed with Equation 21 where K_i represents the stiffness of each rope [23].

$$K = \sum_{i=1}^{n} K_i = K_1 + K_2 + \cdots + K_{n-1} + K_n \tag{21}$$

E.3 Other Key Mechanical Components

Key mechanical components, other than the ropes, specified in this paper are the support columns, the pulleys controlling the ropes in the interconnected rope configuration, the rope mount for the separate rope configuration, and a device that can provide a pre-tensioning effect.

Component	Part Name	Key Specification Measure	Rating	Requirement	Reference
Support columns	Bosch Rexroth Silver Aluminium Profile Strut	Length	3 m	1.6 m	[27]
		Bending Under Maximum Load	0.929 mm (calculated value)	1 mm	
Pulley	Single Pulley Block	Roller Loading	3736N	1139N	[28]
		Maximum rope diameter	12 mm	-	
Pre-tensioning method	CMC CLUTCH	Tension Load	2668N	679N	[29]
		Maximum rope diameter	13 mm	-	
Rope Mount	Wall Ceiling Mount Bracket for Suspension Strap Trainer	Mount Loading	4448N	679N	[30]

Table 11: Key mechanical components with possible off the shelf solutions. Key component specifications are listed along with the requirements computed as part of this paper.

E.4 Bending experienced by the support columns

To confirm the selected support columns would be strong enough, their strength must be tested. A bending limit of 1 mm was set. To analyse this, Equation 22 is used:

$$\delta = \frac{FH_{pull}^3}{6EI}(3L_{min} - H_{pull}) \cdot f_{safe}, \tag{22}$$

F is the maximum load on the pulley/mount calculated previously, H_{pull} is the height of the upper pulley/mount, E is the column material's Young's modulus, I is the second moment of area of the column's cross-section, L_{min} is the minimum length the column would be, and f_{safe} is a safety factor [24]. This result is then multiplied by two for the interconnected rope configuration as two upper pulleys are present near the top of the column. This equation assumes the column is rigidly fixed to the base plate and that the force from the ropes acts as a point load. The force acting on the lower pulleys was assumed to have a negligible effect due to its low mounting height.

Looking at Table 11 it is evident that no column bent beyond 1 mm however the separate ropes configuration saw lower bending displacements compared to the interconnected ropes configuration.

F Key Elements of Matlab Simulation Code

This section only contains the four key code files. More files can be found in the accompanying zip file.

F.1 Main

```
% --- Code used to understand loading in system based on position of child
% diagram of the 3 columbs of the design
%            L/2
%          |----|
%               o <-Front
%              / \
%             /   \
%            /  *  \
%           /       \
%  Left-> o---------o <-Right
%          |---------|
%               L

clc
clearvars
set(groot, 'defaultFigureWindowState', 'maximized') % will default figures to maximise
format longg    % so that scientific notation isnt used (for me less confusing)

%% bring in information from separate files
setBoundary             % get Boundary structure information

%% Assign patient details and create "Child" (and maximum and minimum versions)
Pat_age = 8;                 % Patient age to consider (currently only 8 (max) and 3 (min) work accurately)


% use interpolation so that if another age is given it can be used
Pat_Mass = linearInterp(3, 8, Boundary.pat_wei_min, Boundary.pat_wei_max, Pat_age);
Pat_Hei = linearInterp(3, 8, Boundary.pat_hei_min, Boundary.pat_hei_max, Pat_age);
Pat_waist = linearInterp(3, 8, Boundary.pat_waist_min, Boundary.pat_waist_max, Pat_age);

opposition_strenght_type = 0;   % -1 is very heavy... 0 is more reasonable

% create Child from Patient class. These values are unchanging
Child = Patient(Pat_age, Pat_Hei, Pat_Mass, Pat_waist, opposition_strenght_type);

Child_max = Patient(Boundary.pat_age_max, Boundary.pat_hei_max, Boundary.pat_wei_max, Boundary.pat_waist_max, opposition_strenght_type);
Child_min = Patient(Boundary.pat_age_min, Boundary.pat_hei_min, Boundary.pat_wei_min, Boundary.pat_waist_min, opposition_strenght_type);


%% Key parameters to alter (not including child information. Alter this above)
% ----- What must be altered for individual optimization -----
R_INTERCON = true;          % if the 3 ropes are interconnected or not

% Harness mount location based on percentage of torso height from bottom.
% ie. If you want as high as possible for best stability set to 100
Pat_Har_Loc = 75;   % (%) percentage up the torso harness is mounted (Default is 75%)

% sets the radius the columns are from the center of the participant
```

```
safety_buffer = 0.1;
% based on (largest child) torso length at 40 deg (from upright) + arm length + trunk radius at 40 deg + safety buffer
safe_col_rad = Child_max.t_len*sind(abs(Boundary.lean_f_max)) + Child_max.arm_len + Child_max.t_rad*cosd(abs(Boundary.twist_ccw_max)) + safety_buffer;
col_rad = safe_col_rad;     % (m) selected column radius



% ----- What must be altered for range of locaitons calculation for one individual -----
K_ROPE = 986.97;            % (N/m) spring constant of ropes (assumed all the same)
Pre_Ten = 109.868;          % (N) Pre-tensioning applied to all ropes
UPul_Offset = 0.460202;     % (m) offset of upper pulley height from harness (+ is above, - is below)


% 2D matrix stores positions interested in for analysis (position, value)
% testing 6 "positions" (can change easily)
% 2 lean points of left front and right back
% 3 twist angles of +-45 and 0
positions_of_interest = [0 Boundary.lean_b_max 0;
                         Boundary.lean_l_max 0 0;
                         Boundary.lean_r_max 0 0;
                         Boundary.lean_l_max Boundary.lean_f_max Boundary.twist_ccw_max;
                         Boundary.lean_l_max Boundary.lean_f_max Boundary.twist_cw_max];



% ----- What must be altered if a location is known -----
% Angles related to participant's leaning position
% Pat_lean_medlat = positions_of_interest(1, 1);  % (deg) side to side lean (- = lean left)
% Pat_lean_antpost = positions_of_interest(1, 2);  % (deg) forward backward lean (- = lean forward)
% Pat_twist = positions_of_interest(1, 3);         % (deg) twisting of trunk (- = twist clocwise)

Pat_lean_antpost = Boundary.lean_f_max;
Pat_lean_medlat = Boundary.lean_l_max;
Pat_twist = Boundary.twist_ccw_max;


% create Vars (Variables class)
Vars = Variables(Pat_lean_medlat, Pat_lean_antpost, Pat_twist, Pat_Har_Loc, UPul_Offset, Pre_Ten, K_ROPE, col_rad);



%% Will show the relevant item in the "3D model" of the device
% true indicates you want to show the item the variable name reffers to
flag_SHOW.disp_3d_model = true;
flag_SHOW.disp_text = true;
flag_SHOW.show_field = false;


% visual aids/extras for 3D visualization
```

```
flag_SHOW.start_location = false;            % visibility of phantom starting position
flag_SHOW.main_lables = false;               % visibility of CoM and center of rotation lables
flag_SHOW.extra_lables = false;              % visibility of head and shoulder height lables
flag_SHOW.ref_stars = false;                 % visibility of refference stars for harness (also phantom harness)
flag_SHOW.radius_circle = false;             % visibility of radius of the device indicator
flag_SHOW.net_force_at_center = false;
flag_SHOW.vector_magnitude = false;

% what force vectors to show for all harness points
flag_SHOW.t_net_vec = false;
flag_SHOW.rope_force = false;
flag_SHOW.torque_gen_forces = false;
flag_SHOW.parasitic_forces = false;
flag_SHOW.moments = false;

% if your creating plots for the paper, set this to true so extra information is not included
for_paper = true;


%% ---------- What kind of operation do you want to do? ----------
% 1 = single run
% 2 = run simulation across a range of positions
% 200 = Display cube with coloured dots and arrows
% 201 = Display cube with coloured dots
% 202 = Display cube with arrows
% 220 = Display 1 sphere per twist angle                 (PROBABLY BROKEN)
% 221 = Display sphere for 0 twist
% 222 = Display torque directions in 2D plane per twist angle
% 223 = Display all twist angles in 1 sphere             (PROBABLY BROKEN)
% 225 = Display a force field and sphere plots next to each other
% 25 = Display topographical map
% 3 = genetic optimization for given individual
% 30 = genetic optimization for extremes of individuals and 2 rope configurations (4 different runs)
% 4 = run each permutation for given position            (PROBABLY BROKEN)
% 40 = run each permutation                              (THIS IS VERY SLOW AND MIGHT BE BROKEN)
% 5X = sensitivity analysis across selected variable
%       X = 0: look at interconnected rope effect
%       X = 1: look at rope stiffness effect
%       X = 2: look at column radius effect
%       X = 3: look at upper pulley offset effect
%       X = 4: look at pre tension effect
%       X = 5: look at harness location effect
% 500 = sensitivity analysis across ALL variables (Save figures)
% 501 = sensitivity analysis across ALL variables (Don't save figures)
% 502 = sensitivity analysis but plotted for each variable
flag_run_type = 1;



%% Settings for the different run states
% ===== set precision for created data set =====
```

```
inc_deg = 5;                  % precision of degrees
inc_har = 40;                 % precision of harness height

inc_kro = 200;                % precision of rope stiffness K
inc_ten = 100;                % precision of pre-tension of the ropes
inc_upuloff = 0.2;            % precision of the upper pulley height offset
inc_radius = 0.1;             % precision of the radius from center of child to columns


%===== set precision for created data set (NEW format) =====
num_lean_medlat = 9;    % number of medio-lateral lean points to test (including boundary max/min)
num_lean_antpos = 12;   % number of ant/pstertior lean points to test (including boundary max/min)
num_twist = 3;          % number of twist points to test (including boundary max/min)


%% Set-up data output storage and other small things
output_data_row = 1;        % used to track number of rows of data points
col_green = [0 0.8 0];      % green color


disp("----- Starting simulation -----")
tic;        % starts recording time
disp(datetime)


output_data.patient = Child;



%% options for what to run
% just run the given angles
if flag_run_type == 1
    output_data.sim_results = sim_device(Vars, Child, R_INTERCON, flag_SHOW, Boundary);

else    % if other stuff is to be run, remove the need to print anything in base function
    % changing only these 2 will mean nothing will be printed or otherwise displayed
    flag_SHOW.disp_3d_model = false;
    flag_SHOW.disp_text = false;
end


% run data based visualizations/functions
% display information in a cube
if flag_run_type == 2
    [output_data.sim_results, output_data.vars] = simManyPositions(num_lean_medlat, num_lean_antpos, num_twist, Vars, Child, R_INTERCON, flag_SHOW, Boundary);


elseif 200 <= flag_run_type && flag_run_type <= 202

    % run code for each point
    [output_data.sim_results, output_data.vars, output_data_row] = simManyPositions(num_lean_medlat, num_lean_antpos, num_twist0, Vars, Child, R_INTERCON,
```

```
        flag_SHOW, Boundary);



    for lean_medlat=Boundary.lean_l_max:inc_deg:Boundary.lean_r_max
        % Lean med-lat
        Vars.l_medlat = lean_medlat;
        for lean_antpost=Boundary.lean_f_max:inc_deg:Boundary.lean_b_max
            % Lean ant-post
            Vars.l_antpos = lean_antpost;
            for p_twist=Boundary.twist_cw_max:inc_deg:Boundary.twist_ccw_max
                                % Twist
                Vars.twist = p_twist;

                output_data.sim_results(output_data_row) = sim_device(Vars, Child,
                    R_INTERCON, flag_SHOW, Boundary);
                output_data.vars(output_data_row) = Vars;

                output_data_row = output_data_row+1;
            end
        end
    end

    % create figure and set values
    fig_labs = ["Med-Lat (Deg)" "Ant-Post (Deg)" "Twist (Deg)"];
    fig_lims = [Boundary.lean_l_max Boundary.lean_r_max; Boundary.lean_f_max
        Boundary.lean_b_max; Boundary.twist_cw_max Boundary.twist_ccw_max];
    setup_fig("fig", [0 0 0], false, "ropes", 0, R_INTERCON, [135 60], fig_labs,
        fig_lims, true, false, true);


    scatter3(0, 0, 0, 2000, "red", ".");    % dot indicating 0 0 0


    % needed for the legend to work nicely (dumb work around)
    quiver3(0, 0, 0, 0.1, 0.1, 0.1, "Color", col_green);
    leg_txt1 = "Torque Vectors";
    leg_txt2 = "Torque Magnitude";


    % 2 separate if statements allows for both types of plots to be graphed together

    if flag_run_type == 200 || flag_run_type == 201     % Display cube with coloured
         dots
        sct_d = scatter3(vertcat(output_data.vars(1, :).l_medlat), vertcat(
            output_data.vars(1, :).l_antpos), vertcat(output_data.vars(1, :).twist),
             vertcat(output_data.sim_results.t_net_mag)*20, vertcat(output_data.
            sim_results.t_net_mag), ".");


        % create color bar stuff
        col_bar = colorbar;
        col_bar.Label.String = 'Net Torque (Nm)';

        col_bar_col=get(col_bar,'Limits');
        set(col_bar,'Ticks',linspace(col_bar_col(1),col_bar_col(2),5))
    end

    if flag_run_type == 200 || flag_run_type == 202     % Display cube with arrows
```

```
        % convert forces into the torque
        torq_vects_conv = change_origin([vertcat(output_data.vars(1, :).l_medlat) vertcat(output_data.vars(1, :).l_antpos) vertcat(output_data.vars(1, :).twist)], vertcat(output_data.sim_results.t_net_vec));

        quiv = quiver3(vertcat(output_data.vars(1, :).l_medlat), vertcat(output_data.vars(1, :).l_antpos), vertcat(output_data.vars(1, :).twist), torq_vects_conv(:, 1), torq_vects_conv(:, 2), torq_vects_conv(:, 3), 2);
        quiv.MaxHeadSize = 0.05;
        quiv.LineWidth = 1;
        quiv.Color = col_green;
    end

    % if the arrows wont be printed then remove it from legend by seting text to nothing
    if flag_run_type == 201
        leg_txt1 = "";
    end
    % if the dots wont be printed then remove it from legend by seting text to nothing
    if flag_run_type == 202
        leg_txt2 = "";
    end

    legend("Central position", leg_txt1, leg_txt2, 'Position', [0.65 0.8 0.25 0.1]);


elseif flag_run_type == 220     % Display 1 sphere per twist angle

    inc_deg_twist = 45;
    plot_num = length(Boundary.twist_cw_max:inc_deg_twist:Boundary.twist_ccw_max);

    % trying to make it as square as possible
    plot_cols = ceil(sqrt(plot_num));
    plot_rows = ceil(plot_num/plot_cols);

    plot_count = 1;


    % set up figure nicely
    fig_labs = ["" "" ""];
    fig_lims = [0 0; 0 0; 0 0];
    setup_fig("fig", [0 0 0], false, "ropes", 0, R_INTERCON, [135 60], fig_labs, fig_lims, true, true, true);



    % loop though the different plots and graph stuff untill everything has been done
    for p_twist=Boundary.twist_cw_max:inc_deg_twist:Boundary.twist_ccw_max     % Twist
        Vars.twist = p_twist;

        fig_labs = ["Med-Lat (m)" "Ant-Post (m)" "Height"];
        fig_lims = [0 0; 0 0; 0 0];
        setup_fig("sub", [plot_rows plot_cols plot_count], [1 1 1], "angles", Vars.twist, R_INTERCON, [-150 40], fig_labs, fig_lims, true, true, true);

        %view([180 90])
```

```
        for lean_antpost=Boundary.lean_f_max:inc_deg:Boundary.lean_b_max  % Lean ant-post
            Vars.l_antpos = lean_antpost;
            for lean_medlat=Boundary.lean_l_max:inc_deg:Boundary.lean_r_max  % Lean med-lat
                Vars.l_medlat = lean_medlat;

                output_data.sim_results = sim_device(Vars, Child, R_INTERCON, flag_SHOW, Boundary);
                output_data.vars = Vars;


                % plot points and the line that would emerge from it
                %plot_point(output_data.sim_results.har_c, ".", 5, "b", "", "Left")
                plot_force_vec(output_data.sim_results.har_c, output_data.sim_results(output_data_row).t_net_vec, "-", "b", "", 'left', 0.01)
            end
        end


        % plot central line to show vertical center point
        plot_line([output_data.sim_results.center_xy (Child.t_len + Child.chair)], [output_data.sim_results.center_xy 0], "-", "r")

        % plot rings to show spherical shape
        for ang=Boundary.lean_f_max:10:0
            hei = (Child.t_len*Vars.h_loc/100)*cosd(ang) + Child.chair;
            rad = (Child.t_len*Vars.h_loc/100)*sind(ang);
            plotCircle3D([output_data.sim_results.center_xy hei], [0 0 hei], rad, 'r:')
        end

        plot_count = plot_count+1;  % increment so next plot will be in different subplot
    end

    legend("Central position", "", "Torque Vector direciton", 'Position', [0.62 0.25 0.35 0.2]);


elseif flag_run_type == 221      % Display sphere for 0 twist angle
    twist_ang = 0;
    plotForceFieldHemisphere(0, 1, num_lean_medlat, num_lean_antpos, twist_ang, Vars, Child, R_INTERCON, flag_SHOW, Boundary, "", false, false)


elseif flag_run_type == 222      % Display torque directions in 2D plane per twist angle
    % if sensitivity analysis looks at defined range
    r_k_vals = [30 100 200];          % [low medium high] rope stiffness values
    r_preten_vals = [70 110 150];     % [low medium high] pre tensioning values


    twist_ang = 0;
```

```
    %plot1ForceField(0, [0 0 1], num_lean_medlat, num_lean_antpos, twist_ang, "angles", Vars, Child, R_INTERCON, flag_SHOW, Boundary, '', false, for_paper)
    plotForceFields(num_lean_medlat, num_lean_antpos, num_twist, Vars, Child, R_INTERCON, flag_SHOW, Boundary, '', false, for_paper)
    %plotForceFieldHemisphere(0, 1, num_lean_medlat, num_lean_antpos, twist_ang, Vars, Child, R_INTERCON, flag_SHOW, Boundary, "", false, false)
    %plotSensitivityAnalysis_field(num_lean_medlat, num_lean_antpos, twist_ang, r_k_vals, r_preten_vals, Vars, Child, R_INTERCON, flag_SHOW, Boundary, '', false, for_paper)


elseif flag_run_type == 225     % Display a force field and sphere plots next to each other

    % set up figure nicely
    fig_labs = ["" "" ""];
    fig_lims = [0 0; 0 0; 0 0];
    setup_fig("fig", [0 0 0], false, "ropes", 0, R_INTERCON, [90 0], fig_labs, fig_lims, false, false, false);

    twist_ang = 0;

    plot1ForceField(1, [1 2], num_lean_medlat, num_lean_antpos, twist_ang, "angles", Vars, Child, R_INTERCON, flag_SHOW, Boundary, '', false, for_paper);

    plotForceFieldHemisphere(2, 2, num_lean_medlat, num_lean_antpos, twist_ang, Vars, Child, R_INTERCON, flag_SHOW, Boundary, "", false, false);


elseif flag_run_type == 223     % Display all twist angles in 1 sphere
    inc_deg_twist = 45;
    col_opt = {[1 0 0], col_green, [0 0 0], [1 0 1], [0 0 1]};

    style_count = 1;

    % set up figure nicely
    fig_labs = ["Med-Lat (m)" "Ant-Post (m)" "Height"];
    fig_lims = [0 0; 0 0; 0 0];
    setup_fig("fig", [0 0 0], [1 1 1], "ropes", 0, R_INTERCON, [-150 40], fig_labs, fig_lims, true, true, true);



    for lean_antpost=Boundary.lean_f_max:inc_deg:Boundary.lean_b_max                        % Lean ant-post
        Vars.l_antpos = lean_antpost;
        for lean_medlat=Boundary.lean_l_max:inc_deg:Boundary.lean_r_max                      % Lean med-lat
            Vars.l_medlat = lean_medlat;
            for p_twist=Boundary.twist_cw_max:inc_deg_twist:Boundary.twist_ccw_max           % Twist
                Vars.twist = p_twist;

                %for Vars.twist=90           % Twist
                output_data.sim_results(output_data_row) = sim_device(Vars, Child, R_INTERCON, flag_SHOW, Boundary);
                output_data.vars(output_data_row) = Vars;


```

```
                    % plot points and the line that would emerge from it
                    plot_point(output_data.sim_results(output_data_row).har_c(output_data_row), ".", 5, col_opt{style_count}, "", "Left")
                    plot_force_vec(output_data.sim_results(output_data_row).har_c(output_data_row), output_data.sim_results(output_data_row).t_net_vec, "-", col_opt{style_count}, "", 'left', 2)


                    style_count = style_count+1;                    % increment so colors work
                end
                style_count = 1;    % reset colors
            end
        end

        % plot rings to show spherical shape
        for ang=Boundary.lean_f_max:10:0
            hei = Child.t_len*cosd(ang);
            rad = Child.t_len*sind(ang);
            plotCircle3D([0 0 hei], [0 0 hei], rad, 'r:')
        end

        % plot central line to show vertical center point
        plot_line([0 0 0.5], [0 0 0], "-", "r")

        ledg = legend("", "Clockwise 90 deg", "", "", "Clockwise 45 deg", "", "", "0 deg", "", "", "Anticlockwise 45 deg", "", "", "Anticlockwise 90 deg");
        ledg.Title.String = 'Twist angles';




    elseif flag_run_type == 25      % Display topographical map

        inc_deg_twist = 45;
        plot_num = length(Boundary.twist_cw_max:inc_deg_twist:Boundary.twist_ccw_max);

        % trying to make it as square as possible
        plot_cols = ceil(sqrt(plot_num));
        plot_rows = ceil(plot_num/plot_cols);

        plot_count = 1;


        % set up figure nicely
        fig_labs = ["" "" ""];
        fig_lims = [0 0; 0 0; 0 0];
        fig_inf = setup_fig("fig", [0 0 0], false, "ropes", 0, R_INTERCON, [90 0], fig_labs, fig_lims, false, false, false);

        % starting min and max torque mag values
        t_max = 0;
        t_min = 5;


        % loop though the different plots and graph stuff untill everything has been done
        for p_twist=Boundary.twist_cw_max:inc_deg_twist:Boundary.twist_ccw_max                                  % Twist
```

```
        Vars.twist = p_twist;

        len_x = length(Boundary.lean_f_max:inc_deg:Boundary.lean_b_max);
        len_y = length(Boundary.lean_l_max:inc_deg:Boundary.lean_r_max);

        for lean_antpost=Boundary.lean_f_max:inc_deg:Boundary.lean_b_max
            % Lean ant-post
            Vars.l_antpos = lean_antpost;
            for lean_medlat=Boundary.lean_l_max:inc_deg:Boundary.lean_r_max
                                % Lean med-lat
                Vars.l_medlat = lean_medlat;

                output_data.sim_results(output_data_row) = sim_device(Vars, Child,
                    R_INTERCON, flag_SHOW, Boundary);
                output_data.vars(output_data_row) = Vars;

                output_data_row = output_data_row+1;
            end
        end

        % used to set size of color bar
        if t_max < max(vertcat(output_data.sim_results.t_net_vec))
            t_max = max(vertcat(output_data.sim_results.t_net_vec));
        end
        if t_min > min(vertcat(output_data.sim_results.t_net_vec))
            t_min = min(vertcat(output_data.sim_results.t_net_vec));
        end

        % set up sub-figure nicely put here so we can set vertical limits as the
            same
        fig_labs = ["Med-Lat (deg)" "Ant-Post (deg)" "Torque Magnitude (Nm)"];
        fig_lims = [Boundary.lean_l_max Boundary.lean_r_max; Boundary.lean_f_max
            Boundary.lean_b_max; 0 0];
        fig_inf = setup_fig("sub", [plot_rows plot_cols plot_count], [1 1 0], "
            angles", Vars.twist, R_INTERCON, [-150 40], fig_labs, fig_lims, true,
            true, true);


        % https://uk.mathworks.com/matlabcentral/answers/712893-why-surface-plot-is-
            connecting-its-edges
        % idk wby exactly griddata messes things up but below works and above does
            weird stuff
        D = [vertcat(output_data.vars(1, vertcat(output_data.vars(1, :).twist) ==
            Vars.twist).l_medlat), vertcat(output_data.vars(1, vertcat(output_data.
            vars(1, :).twist) == Vars.twist).l_antpos), vertcat(output_data.
            sim_results(vertcat(output_data.vars(1, :).twist) == Vars.twist).
            t_net_mag)];
        [notused, ix] = unique(D(:,2));                         % Determine Initial
            Indices Of Repeated Elements
        ixd = diff(ix);                                         % Lenmgth Of
            Repeated Elements
        Dr = reshape(D,mean(ixd),[]);                           % Reshape Data
            Matrix
        collen = size(Dr,2)/size(D,2);                          % Column Block
            Lengths

        for k = 1:size(D,2)
            data_trim{k} = Dr(:,(1:collen)+(k-1)*collen);               % Segment By
                Column Blocks
```

```
        end

        surf(data_trim{1}, data_trim{2}, data_trim{3})      % plot corrected data

        % just to be sure things are working right
        % scatter3(output_data(output_data.vars(1, :).twist == Vars.twist, 2)], [output_data(output_data.vars(1, :).twist == Vars.twist, 3)], [output_data(output_data.vars(1, :).twist == Vars.twist, 1))


        plot_line([0 0 0], [0 0 20], "-", "r")
        %red_dot = scatter3(0, 0, 10, 1000, "red", ".");    % dot indicating 0 0 0

        plot_count = plot_count+1;  % increment so next plot will be in different subplot

        %view([180, 90])

    end

    % create color bar stuff
    col_bar = colorbar(fig_inf, 'Position',[0.94 0.168 0.022 0.7]);
    col_bar.Label.String = 'Net Torque (Nm)';
    %clim(fig_inf, [t_min, t_max]);       % set colorbar limits


    %subaxis(plot_rows, plot_cols, plot_count, 'Spacing', 0.08);
    %plot_line([0 0 0], [0 0 20], "-", "r")
    legend("", "Central position", 'Position', [0.75 0.3 0.1 0.07])



% genetic optimization algorythm
% https://www.youtube.com/watch?v=uQj5UNhCPuo
elseif flag_run_type == 3 || flag_run_type == 30
    % changing setup of function
    size_of_pop = 100;              % total number of individuals in population
    num_top_sol = size_of_pop/10;   % number of top solutions saved
    num_fail_gen = 100;             % number of generaions that can pass until algorythm stops
    size_mutation = 10;             % maximum % change mutation can cause
    num_of_repeat_attem = 10;       % how many times to repeat testing to ensure output is stable


    output_data_row = 1;            % for keeping track of number of stuff


    % 75 was selected as a good middle ground. A therapist could change this themselves to suit training needs
    % using calculated value for safe column radius
    % setting both here to ensure it consistency
    Vars.h_loc = 75;
    Vars.rad_col = safe_col_rad;

    % set values needed for plotting
    l_medlat_ang = positions_of_interest(1, 1);
    l_antpos_ang = positions_of_interest(1, 2);
    twist_ang = positions_of_interest(1, 3);
```

```
    har_resolution = 25;        % (unit) number of points to look at per variable (resolution)


    % look though the changing harness height to ensure everything looks ok
    var_change = 5;
    var_range = linspace(Boundary.harn_h_min, Boundary.harn_h_max, har_resolution);

    if flag_run_type == 30
        Child_TEST = [Child_min Child_max];
        R_CONFIGS = [true false];
    else
        Child_TEST = Child;
        R_CONFIGS = R_INTERCON;
    end


    % find optimal solutions for an individual child
    if flag_run_type == 3 || flag_run_type == 30

        num_sims = length(Child_TEST)*length(R_CONFIGS);    % number of simulations to be completed
        optimization_criteria_count = 1;

        % run the optimization to get all the data outputs
        for r_conf=1:length(R_CONFIGS)    % loop though different rope configurations
            for chil=1:length(Child_TEST)       % loop though different children to test

                track_time_elapsed("Starting optimization version", optimization_criteria_count, num_sims)

                % save information about the child being tested and rope configuration
                output_data(r_conf, chil).patient = Child_TEST(chil);
                output_data(r_conf, chil).r_config = R_CONFIGS(r_conf);


                for i=1:num_of_repeat_attem
                    track_time_elapsed("  Starting genetic optimization attempt", i, num_of_repeat_attem)

                    % run the genetic optimization to return an optimized result for given child parameters
                    [output_data(r_conf, chil).gen_optim(i), flags_Optim(r_conf, chil, i)] = optim_genetic_with_cost_func(Vars, Child_TEST(chil), R_CONFIGS(r_conf), flag_SHOW, Boundary, positions_of_interest, size_of_pop, num_top_sol, num_fail_gen, size_mutation);

                    Vars.r_k = output_data(r_conf, chil).gen_optim(i).r_k;
                    Vars.r_preten = output_data(r_conf, chil).gen_optim(i).r_preten;
                    Vars.p_offset = output_data(r_conf, chil).gen_optim(i).p_offset;

                    % go though each position of interest
                    for poi=1:size(positions_of_interest, 1)
                        Vars.l_medlat = positions_of_interest(poi, 1);
                        Vars.l_antpos = positions_of_interest(poi, 2);
```

```
                        Vars.twist = positions_of_interest(poi, 3);

                        % Run the simulation independently to save its results
                        output_data(r_conf, chil).sim_results(i, poi) = sim_device(Vars, Child_TEST(chil), R_CONFIGS(r_conf), flag_SHOW, Boundary);
                    end

                    % find position of interest with:
                    % maximum torque value
                    [~, poi_max_t(r_conf, chil, i)] = max([output_data(r_conf, chil).sim_results(i, :).t_net_mag]);
                    % maximum tension value
                    [~, poi_max_ten(r_conf, chil, i)] = max([output_data(r_conf, chil).sim_results(i, :).f_mag_rope_max]);
                    % min torque value
                    [~, poi_min_t(r_conf, chil, i)] = min([output_data(r_conf, chil).sim_results(i, :).t_net_mag]);
                    % worst torque direction value
                    [~, poi_worst_t_direc(r_conf, chil, i)] = max([output_data(r_conf, chil).sim_results(i, :).t_net_direc]);
                    % highest parasitic force
                    [~, poi_max_f_para(r_conf, chil, i)] = max([output_data(r_conf, chil).sim_results(i, :).f_mag_para_max]);
                    fprintf("\n")
                end

                % find best (min) solution based on cost function
                [~, top_n_indiv(r_conf, chil)] = min([output_data(r_conf, chil).gen_optim.v_cost_val]);

                optimization_criteria_count = optimization_criteria_count+1;    % purely for time keeping
                fprintf("\n")   % formating
            end
        end


        % save the output_data variable
        data_save_name = strcat(pwd, "\Optimization results\all_output_data.mat");

        name_bad = true;
        name_increment = 0;

        while name_bad
            % check if duplicates exist
            if ~exist(data_save_name)
                % file does not exist, create it and end loop
                save(data_save_name, "output_data", "flags_Optim", "top_n_indiv", "poi_max_t", "poi_max_ten", "poi_min_t", "poi_worst_t_direc", "poi_max_f_para")
                name_bad = false;

            else
                % file already exists, change name
                name_increment=name_increment+1;
                data_save_name = strcat(pwd, "\Optimization results\all_output_data_(", num2str(name_increment), ").mat");
            end
```

```matlab
        end


        %% Run loop again to generate text files + table
        % can run from here if optimization data is given and first part of section is run till
        % first optimization begins
        situation_count = 1;

        for r_conf=1:length(R_CONFIGS)     % loop though different rope configurations
            for chil=1:length(Child_TEST)       % loop though different children to test

                track_time_elapsed("Saving text file of run", situation_count, num_sims)

                % set to best values
                Vars.r_k = output_data(r_conf, chil).gen_optim(top_n_indiv(r_conf, chil)).r_k;
                Vars.r_preten = output_data(r_conf, chil).gen_optim(top_n_indiv(r_conf, chil)).r_preten;
                Vars.p_offset = output_data(r_conf, chil).gen_optim(top_n_indiv(r_conf, chil)).p_offset;


                % used to identify this optimization run
                plot_name = strcat('Opti_res_', num2str(Vars.rad_col), 'm_', num2str(Vars.h_loc), '_', string(R_CONFIGS(r_conf)), "_", num2str(Child_TEST(chil).age), "YO");


                % this loop here purely so that two different files can be saved. One that is a bit
                % easier to read for people and another that can easily be coppied into a table in
                % latex
                for save_ver=1:2
                    % --- save the information to a file

                    % create the file that will hold the base values for the executed sensitivity analysis
                    % name based on fixed properties
                    if save_ver == 2
                        file_name = create_file("\Optimization results", strcat("LATEX_", plot_name), ".txt");
                    else
                        file_name = create_file("\Optimization results", plot_name, ".txt");
                    end

                    % create text to inform on rope settings based
                    if R_CONFIGS(r_conf)
                        rope_setup_text = 'Connected';
                    else
                        rope_setup_text = 'Separate';
                    end

                    % If the file doesn't exist, create it with write permission
```

```
fileID = fopen(file_name,'w');

fprintf(fileID, "\t---Results of genetic optimization - optimal value (mean)[median]{SD} <units>---\n\n");
fprintf(fileID, "Unchanging properties are:\n");
fprintf(fileID, "\tColumn Radius = %g <m>\n", Vars.rad_col);
fprintf(fileID, "\tHarness height on torso = %g <%%>\n", Vars.h_loc);
fprintf(fileID, strcat("\tRope Configuration: ", rope_setup_text, "\n"));
fprintf(fileID, "\tPatient Age = %g <Years>\n", Child_TEST(chil).age);
fprintf(fileID, "\tChild Opposition Strength = %g <Nm>\n", Child_TEST(chil).t_opp_str);
fprintf(fileID, "\n\nMechanical properties are:\n");
print_opti_info(fileID, [output_data(r_conf, chil).gen_optim.r_k], top_n_indiv(r_conf, chil), "Rope Stiffness", "N/m", save_ver)
print_opti_info(fileID, [output_data(r_conf, chil).gen_optim.r_preten], top_n_indiv(r_conf, chil), "Pre-tensioning", "N", save_ver)
print_opti_info(fileID, [output_data(r_conf, chil).gen_optim.p_offset], top_n_indiv(r_conf, chil), "Pulley offset", "m", save_ver)
fprintf(fileID, "\n\nMachine performance data for best individual:\n");
print_opti_info(fileID, [output_data(r_conf, chil).sim_results(:, poi_max_t(r_conf, chil, top_n_indiv(r_conf, chil))).t_net_mag], top_n_indiv(r_conf, chil), "Highest torque magnitude", "Nm", save_ver)
print_opti_info(fileID, [output_data(r_conf, chil).sim_results(:, poi_max_ten(r_conf, chil, top_n_indiv(r_conf, chil))).f_mag_rope_max], top_n_indiv(r_conf, chil), "Highest tension in ropes", "N", save_ver)
print_opti_info(fileID, [output_data(r_conf, chil).sim_results(:, poi_min_t(r_conf, chil, top_n_indiv(r_conf, chil))).t_net_mag], top_n_indiv(r_conf, chil), "Inner limit torque magnitude", "Nm", save_ver)
print_opti_info(fileID, [output_data(r_conf, chil).sim_results(:, poi_worst_t_direc(r_conf, chil, top_n_indiv(r_conf, chil))).t_net_direc], top_n_indiv(r_conf, chil), "Worst torque direction", "deg", save_ver)
print_opti_info(fileID, [output_data(r_conf, chil).sim_results(:, poi_max_f_para(r_conf, chil, top_n_indiv(r_conf, chil))).f_mag_para_max], top_n_indiv(r_conf, chil), "Maximum parasitic force", "N", save_ver)
fprintf(fileID, "\n\nOptmization data:\n");
print_opti_info(fileID, [output_data(r_conf, chil).gen_optim.num_gen_best], top_n_indiv(r_conf, chil), "Generations required", "num", save_ver)
print_opti_info(fileID, [output_data(r_conf, chil).gen_optim.v_cost_val], top_n_indiv(r_conf, chil), "V Cost", "units", save_ver)
fprintf(fileID, "\n\nOptmization logic tracking:\n");
print_opti_info(fileID, [flags_Optim(r_conf, chil, :).tdirec], top_n_indiv(r_conf, chil), "Torque direction tracking", "", save_ver)
print_opti_info(fileID, [flags_Optim(r_conf, chil, :).tmag], top_n_indiv(r_conf, chil), "Torque magnitude tracking", "",
```

```
save_ver)
print_opti_info(fileID, [flags_Optim(r_conf, chil, :).flg_check], top_n_indiv(r_conf, chil), "Checking various flags", "", save_ver)
print_opti_info(fileID, [flags_Optim(r_conf, chil, :).converged], top_n_indiv(r_conf, chil), "Convergence of the optimization", "Bool", save_ver)
fprintf(fileID, "\n\nSafety Checks:\n");
print_opti_info(fileID, [output_data(r_conf, chil).sim_results(:, poi_max_t(r_conf, chil, top_n_indiv(r_conf, chil))).r_strch_perc_0], top_n_indiv(r_conf, chil), "Starting rope stretch", "%", save_ver)
print_opti_info(fileID, [output_data(r_conf, chil).sim_results(:, poi_max_t(r_conf, chil, top_n_indiv(r_conf, chil))).r_strch_perc_max], top_n_indiv(r_conf, chil), "Maximum rope stretch", "%", save_ver)
print_opti_info(fileID, [output_data(r_conf, chil).sim_results(:, poi_max_t(r_conf, chil, top_n_indiv(r_conf, chil))).flg_r_unstrchlen], top_n_indiv(r_conf, chil), "Unstretched rope good", "Bool", save_ver)
print_opti_info(fileID, [output_data(r_conf, chil).sim_results(:, poi_max_t(r_conf, chil, top_n_indiv(r_conf, chil))).flg_r_strchprop_0], top_n_indiv(r_conf, chil), "Initial rope stretch good", "Bool", save_ver)
print_opti_info(fileID, [output_data(r_conf, chil).sim_results(:, poi_max_t(r_conf, chil, top_n_indiv(r_conf, chil))).flg_trunk_len], top_n_indiv(r_conf, chil), "Trunk length good", "Bool", save_ver)
print_opti_info(fileID, [output_data(r_conf, chil).sim_results(:, poi_max_t(r_conf, chil, top_n_indiv(r_conf, chil))).flg_no_slk_lr], top_n_indiv(r_conf, chil), "No slack in LR rope", "Bool", save_ver)
print_opti_info(fileID, [output_data(r_conf, chil).sim_results(:, poi_max_t(r_conf, chil, top_n_indiv(r_conf, chil))).flg_no_slk_rf], top_n_indiv(r_conf, chil), "No slack in RF rope", "Bool", save_ver)
print_opti_info(fileID, [output_data(r_conf, chil).sim_results(:, poi_max_t(r_conf, chil, top_n_indiv(r_conf, chil))).flg_no_slk_fl], top_n_indiv(r_conf, chil), "No slack in FL rope", "Bool", save_ver)
print_opti_info(fileID, [output_data(r_conf, chil).sim_results(:, poi_max_t(r_conf, chil, top_n_indiv(r_conf, chil))).flg_r_strchprop_lr_end], top_n_indiv(r_conf, chil), "Final rope stretch good in rope LR", "Bool", save_ver)
print_opti_info(fileID, [output_data(r_conf, chil).sim_results(:, poi_max_t(r_conf, chil, top_n_indiv(r_conf, chil))).flg_r_strchprop_rf_end], top_n_indiv(r_conf, chil), "Final rope stretch good in rope RF", "Bool", save_ver)
print_opti_info(fileID, [output_data(r_conf, chil).sim_results(:, poi_max_t(r_conf, chil, top_n_indiv(r_conf, chil))).flg_r_strchprop_fl_end], top_n_indiv(r_conf, chil), "Final rope stretch good in rope FL", "Bool", save_ver)

fclose(fileID);


% now save relevant info into table for easier display to others
fileID = fopen(file_name, 'r');     % Open the file again for reading
```

```
    row_num = 0;    % tracks number of rows saved

    % Read data line by line until the end
    while ~feof(fileID)
        line = fgetl(fileID); % Read one line
        equal_pos = strfind(line, '='); % Find position of "="

        if ~isempty(equal_pos) % If "=" is found
            cut_at_equ = line(equal_pos+1:end);     % Extract the part after "="
            paren_pos = strfind(cut_at_equ, '(');   % Find position of "("

            if isempty(paren_pos)
                paren_pos = strfind(cut_at_equ, '<');   % Find position of "("
            end


            ang1_pos = strfind(cut_at_equ, '<');   % Find position of "<"
            % done in 2 stages to that if there is a ( before it wont mess things up

            num = str2double(cut_at_equ(1:paren_pos-1)); % Convert to a number

            if ~isnan(num) % Check if it's a valid number
                row_num = row_num+1;
                output_table_data(r_conf, chil, row_num) = num;

                % this doesn't work for some reason when 'years' and I have no idea why
                % unit = convertCharsToStrings(cut_at_equ(ang1_pos+1:end-1));        % find unit
                % output_table_units(row_num) = unit;
            end
        end
    end

    fclose(fileID);         % Close the file
end

if flag_run_type == 3
    % print info about best, mean, SD, and all results to easilly see whats going on
    fprintf("\n\n\t\t---Results of genetic optimization - optimal value (mean)[median]{SD} (units)---\n\nMechanical properties are:\n")
    print_opti_info(0, [output_data(r_conf, chil).gen_optim.r_k], top_n_indiv(r_conf, chil), "Rope Stiffness", "N/m", 1)
    print_opti_info(0, [output_data(r_conf, chil).gen_optim.r_preten], top_n_indiv(r_conf, chil), "Pre-tensioning", "N", 1)
    print_opti_info(0, [output_data(r_conf, chil).gen_optim.p_offset], top_n_indiv(r_conf, chil), "Pulley offset", "m", 1)

    fprintf("\nMachine performance data:\n")
```

```
                    print_opti_info(0, [output_data(r_conf, chil).sim_results(:, poi_max_t(r_conf, chil, top_n_indiv(r_conf, chil))).t_net_mag], top_n_indiv(r_conf, chil), "Highest torque magnitude", "Nm", 1)
                    print_opti_info(0, [output_data(r_conf, chil).sim_results(:, poi_max_t(r_conf, chil, top_n_indiv(r_conf, chil))).t_net_direc], top_n_indiv(r_conf, chil), "Torque direction", "units", 1)
                    print_opti_info(0, [output_data(r_conf, chil).sim_results(:, poi_max_t(r_conf, chil, top_n_indiv(r_conf, chil))).f_mag_para_max], top_n_indiv(r_conf, chil), "Maximum parasitic force", "N", 1)

                    fprintf("\nOptmization data:\n")
                    print_opti_info(0, [output_data(r_conf, chil).gen_optim.num_gen_best], top_n_indiv(r_conf, chil), "Generations required", "num", 1)
                    print_opti_info(0, [output_data(r_conf, chil).gen_optim.v_cost_val], top_n_indiv(r_conf, chil), "V Cost", "units", 1)
                end
            end
            situation_count = situation_count+1;
        end

        % save all info into a table
        % create the table so that information can also be put into this
        row_lbls = ["Column Radius" "Harness height" "Age" "Opposition Strength" "K" ...
            "Pre-tension" "Pulley offset" "Highest torque magnitude" "Highest tension in ropes"...
            "Inner limit torque magnitude" "Torque direction" "Maximum parasitic force" ...
            "Generations required" "V Cost" "Torque direction tracking" "Torque magnitude tracking"...
            "Flag tracking" "Convergence of the optimization" "Initial elongation of rope" "Maximum elongation of rope" "Unstretched rope good" ...
            "Initial rope stretch good" "Trunk length good" "No slack in LR rope" ...
            "No slack in RF rope" "No slack in FL rope" "Final rope stretch good in rope LR"...
            "Final rope stretch good in rope RF" "Final rope stretch good in rope FL"]';
        output_table = table(row_lbls, squeeze(output_table_data(1, 1, :)), ...
            squeeze(output_table_data(1, 2, :)), squeeze(output_table_data(2, 1, :)), ...
            squeeze(output_table_data(2, 2, :)), ...
            'VariableNames', ["Vars", "3YO Interconnected", "8YO Interconnected", ...
            "3YO Separate", "8YO Separate"]);

        file_name = create_file("\Optimization results", 'Opti_res_all_output', ".xlsx");

        writetable(output_table, file_name)



        %% Run loop again to generate plots
        % can run from here if optimization data is given and first part of section is run till
```

```
        % first optimization begins
        situation_count = 1;

        % determine how the force field plot should be run
        if flag_run_type == 3
            should_close = false;
        elseif flag_run_type == 30
            should_close = true;
        end

        for r_conf=1:length(R_CONFIGS)    % loop though different rope configurations
            for chil=1:length(Child_TEST)       % loop though different children to test

                track_time_elapsed("Ploting data of run", situation_count, num_sims)

                % set to best values
                Vars.r_k = output_data(r_conf, chil).gen_optim(top_n_indiv(r_conf, chil)).r_k;
                Vars.r_preten = output_data(r_conf, chil).gen_optim(top_n_indiv(r_conf, chil)).r_preten;
                Vars.p_offset = output_data(r_conf, chil).gen_optim(top_n_indiv(r_conf, chil)).p_offset;


                % used to identify this optimization run
                plot_name = strcat('Opti_res_', num2str(Vars.rad_col), 'm_', num2str(Vars.h_loc), '_', string(R_CONFIGS(r_conf)), "_", num2str(Child_TEST(chil).age), "YO");


                % plot and save the force field for the given result
                plotForceFields(num_lean_medlat, num_lean_antpos, num_twist, Vars, Child_TEST(chil), R_CONFIGS(r_conf), flag_SHOW, Boundary, plot_name, should_close, for_paper);


                % plot larger, sensitivity graphs
                name_force_plot_indiv = strcat('Opti_sens_force_indiv_', num2str(Vars.rad_col), 'm_', num2str(Vars.h_loc), '_', string(R_CONFIGS(r_conf)), '_', num2str(Child_TEST(chil).age), 'yo');
                name_graph_plot_indiv = strcat('Opti_sens_graph_indiv_', num2str(Vars.rad_col), 'm_', num2str(Vars.h_loc), '_', string(R_CONFIGS(r_conf)), '_', num2str(Child_TEST(chil).age), 'yo');

                range_multiplyer = 0.2;     % how much above and below each variable the range should be (20%)

                % get range of rope stiffness values
                r_k_val = output_data(r_conf, chil).gen_optim(top_n_indiv(r_conf, chil)).r_k;
                r_k_vals_indiv = [r_k_val*(1-range_multiplyer) r_k_val r_k_val*(1+range_multiplyer)];

                % get range of pre-tension values
                r_preten_val = output_data(r_conf, chil).gen_optim(top_n_indiv(r_conf, chil)).r_preten;
```

```
            r_preten_vals_indiv = [r_preten_val*(1-range_multiplyer) r_preten_val r_preten_val*(1+range_multiplyer)];

            % set the limits of the plots based on child's strength
            y_lims = [-5 Child_TEST(chil).t_opp_str*1.5];

            % plot infromation for each individual
            plotSensitivityAnalysis_field(num_lean_medlat, num_lean_antpos, twist_ang, r_k_vals_indiv, r_preten_vals_indiv, Vars, Child_TEST(chil), R_CONFIGS(r_conf), flag_SHOW, Boundary, name_force_plot_indiv, should_close, for_paper);
            plotSensitivityAnalysis_1graphs_over(l_medlat_ang, l_antpos_ang, twist_ang, r_k_vals_indiv, r_preten_vals_indiv, var_change, var_range, y_lims, Vars, Child_TEST(chil), R_CONFIGS(r_conf), flag_SHOW, Boundary, name_graph_plot_indiv, should_close, for_paper);

            situation_count = situation_count+1;
        end

        % create plots for all the things I can across the range of children
        if flag_run_type == 30
            name_force_plot1 = strcat('Opti_sens_force_between_', num2str(Vars.rad_col), 'm_', num2str(Vars.h_loc), '_', string(R_CONFIGS(r_conf)), '_', num2str(Child_TEST(1).age), 'yo');
            name_force_plot2 = strcat('Opti_sens_force_between_', num2str(Vars.rad_col), 'm_', num2str(Vars.h_loc), '_', string(R_CONFIGS(r_conf)), '_', num2str(Child_TEST(2).age), 'yo');
            name_graph_plot_multi = strcat('Opti_sens_graph_between_', num2str(Vars.rad_col), 'm_', string(R_CONFIGS(r_conf)));

            % what to show for rope stiffness [min mean max]
            r_k_mean = mean([output_data(r_conf, 1).gen_optim(top_n_indiv(r_conf, 1)).r_k output_data(r_conf, end).gen_optim(top_n_indiv(r_conf, end)).r_k]);
            r_k_vals_mean = [output_data(r_conf, 1).gen_optim(top_n_indiv(r_conf, 1)).r_k r_k_mean output_data(r_conf, end).gen_optim(top_n_indiv(r_conf, end)).r_k];

            % what to show for pre-tension [min mean max]
            r_preten_mean = mean([output_data(r_conf, 1).gen_optim(top_n_indiv(r_conf, 1)).r_preten output_data(r_conf, end).gen_optim(top_n_indiv(r_conf, end)).r_preten]);
            r_preten_vals_mean = [output_data(r_conf, 1).gen_optim(top_n_indiv(r_conf, 1)).r_preten r_preten_mean output_data(r_conf, end).gen_optim(top_n_indiv(r_conf, end)).r_preten];

            % set the limits of the plots (based on Child_TEST(2) since they will be stronger)
            y_lims = [-5 Child_TEST(2).t_opp_str*1.5];

            % make the plots based on the means
            plotSensitivityAnalysis_field(num_lean_medlat, num_lean_antpos, twist_ang, r_k_vals_mean, r_preten_vals_mean, Vars, Child_TEST(1), R_CONFIGS(r_conf), flag_SHOW, Boundary, name_force_plot1, should_close, for_paper);
            plotSensitivityAnalysis_field(num_lean_medlat, num_lean_antpos, twist_ang, r_k_vals_mean, r_preten_vals_mean, Vars, Child_TEST(2), R_CONFIGS(r_conf), flag_SHOW, Boundary, name_force_plot2,
```

```
                    should_close, for_paper);
                plotSensitivityAnalysis_9graphs(l_medlat_ang, l_antpos_ang, twist_ang, r_k_vals_mean, r_preten_vals_mean, var_change, var_range, y_lims, Vars, Child_TEST, R_CONFIGS(r_conf), flag_SHOW, Boundary, name_graph_plot_multi, should_close, for_paper);
            end
        end
    end

elseif flag_run_type == 4 || flag_run_type == 40
    % find number of descrete points for each value
    NUM_INCREMENT_KRO = length(Boundary.r_k_min:inc_kro:Boundary.r_k_max);
    NUM_INCREMENT_TEN = length(Boundary.r_preten_min:inc_har:Boundary.r_preten_max);
    NUM_INCREMENT_UPH = length(Boundary.p_offset_min:inc_upuloff:Boundary.p_offset_max);

    % find all results for given position
    if flag_run_type == 4
        % find total number of data points, used for matrix pre-alocaiton
        OUTPUT_LENGTH = NUM_INCREMENT_KRO*NUM_INCREMENT_TEN*NUM_INCREMENT_UPH;

        for k_rope=Boundary.r_k_min:inc_kro:Boundary.r_k_max
            Vars.r_k = k_rope;
            for pre_ten=Boundary.r_preten_min:inc_ten:Boundary.r_preten_max
                Vars.r_preten = pre_ten;
                for pul_offset=Boundary.p_offset_min:inc_upuloff:Boundary.p_offset_max
                    Vars.p_offset = pul_offset;

                    output_data.sim_results(output_data_row) = sim_device(Vars, Child, R_INTERCON, flag_SHOW, Boundary);
                    output_data.vars(output_data_row) = Vars;

                    output_data_row = output_data_row+1;
                end
            end
        end



    % find all results for ALL positions (THIS WILL TAKE A WHILE/MIGHT NOT WORK)
    elseif flag_run_type == 40
        % get number of descrete points for each position
        NUM_INCREMENT_MEDLAT = length(Boundary.lean_l_max:inc_deg:Boundary.lean_r_max);
        NUM_INCREMENT_ANTPOST = length(Boundary.lean_f_max:inc_deg:Boundary.lean_b_max);
        NUM_INCREMENT_ROT = length(Boundary.twist_cw_max:inc_deg:Boundary.twist_ccw_max);
        NUM_INCREMENT_HAR = length(Boundary.harn_h_min:inc_har:Boundary.harn_h_max);

        % find total number of data points, used for matrix pre-alocaiton
        OUTPUT_LENGTH = NUM_INCREMENT_MEDLAT*NUM_INCREMENT_ANTPOST*NUM_INCREMENT_ROT*NUM_INCREMENT_HAR*NUM_INCREMENT_KRO*NUM_INCREMENT_TEN*NUM_INCREMENT_UPH;

        % run code for each permutation
```

```
        for lean_medlat=Boundary.lean_l_max:inc_deg:Boundary.lean_r_max                 % Lean med-lat
            Vars.l_medlat = lean_medlat;
            for lean_antpost=Boundary.lean_f_max:inc_deg:Boundary.lean_b_max             % Lean ant-post
                Vars.l_antpos = lean_antpost;
                for p_twist=Boundary.twist_cw_max:inc_deg:Boundary.twist_ccw_max             % Twist
                    Vars.twist = p_twist;
                    for har_loc=Boundary.harn_h_min:inc_har:Boundary.harn_h_max          % harness location
                        Vars.h_loc = har_loc;
                        for k_rope=Boundary.r_k_min:inc_kro:Boundary.r_k_max                 % K of rope
                            Vars.r_k = k_rope;
                            for pre_ten=Boundary.r_preten_min:inc_ten:Boundary.r_preten_max       % pre-tension
                                Vars.r_preten = pre_ten;
                                for pul_offset=Boundary.p_offset_min:inc_upuloff:Boundary.p_offset_max % upper pulley offset
                                    Vars.p_offset = pul_offset;

                                    output_data.sim_results(output_data_row) = sim_device(Vars, Child, R_INTERCON, flag_SHOW, Boundary);
                                    output_data.vars(output_data_row) = Vars;


                                    output_data_row = output_data_row+1;
                                end
                            end
                        end
                    end
                end
            end
        end
    end


    % used to examin the effect different variables have on the system as a whole
elseif fix(flag_run_type/10) == 5 || flag_run_type == 500 || flag_run_type == 501

    inc_resolution = 10;        % (unit) number of points to look at per variable (resolution)

    % needed for plot verison
    twist_ang = 0;
    r_k_vals = [30 100 200];          % [low medium high] rope stiffness values
    r_preten_vals = [70 110 150];     % [low medium high] pre tensioning values


    if fix(flag_run_type/10) == 5       % only run for one variable
        range = flag_run_type;

    else                                % run for all variables
        range = [50 51 52 53 54 55];
    end


```

```
    if flag_run_type == 500
        % create the file that will hold the base values for the executed
            sensitivity analysis
        file_name = 'base_values';

        file_name = create_file("\Figures", file_name, ".txt");

        if exist(file_name, 'file') == 2
            % If the file exists, open it with append permission
            fileID = fopen(file_name,'a');
        else
            % If the file doesn't exist, create it with write permission
            fileID = fopen(file_name,'w');
        end

        % Write child specific variables
        fprintf(fileID, 'Child age: %g\n', Child.age);
        fprintf(fileID, 'Child height: %g\n', Child.hei);
        fprintf(fileID, 'Child weight: %g\n', Child.wei);
        fprintf(fileID, 'Child trunk radius: %g\n', Child.t_rad);

        % Write set-up specific variables
        fprintf(fileID, 'col_rad_: %g\n', Vars.rad_col);
        fprintf(fileID, 'harn_hei_: %g\n', Vars.h_loc);
        fprintf(fileID, 'r_k_: %g\n', Vars.r_k);
        fprintf(fileID, 'r_preten_: %g\n', Vars.r_preten);
        fprintf(fileID, 'upper_pull_offset_: %g\n', Vars.p_offset);

        % Close the file
        fclose(fileID);
    end



    % this loop is only relevant if user wants to loop though all possible variables
    for flag_analisis_type = range

        % reset variables to original value when looping (R_INTERCON doesn't need to
            be reset)
        Vars.r_k = K_ROPE;
        Vars.rad_col = col_rad;
        Vars.p_offset = UPul_Offset;
        Vars.r_preten = Pre_Ten;
        Vars.h_loc = Pat_Har_Loc;


        if flag_analisis_type == 50      % look at interconnected rope effect
            for i=[1 2]
                fprintf('\n\n')
                track_time_elapsed("Running rope config point", i, 2)
                fprintf('\n')

                % set interconnection value
                if i == 1
                    R_INTERCON_new = true;

                else
                    R_INTERCON_new = false;

```

```
                end


                % run simulation and save result
                output_data.sim_results(i) = sim_device(Vars, Child, R_INTERCON_new, flag_SHOW, Boundary);


                if flag_run_type == 500
                    % when the file should be saved
                    file_name = strcat("r_intercon_", string(R_INTERCON_new));
                else
                    file_name = "";
                end


                plotForceFields(num_lean_medlat, num_lean_antpos, num_twist, Vars, Child, R_INTERCON_new, flag_SHOW, Boundary, file_name, true , for_paper)
                plotSensitivityAnalysis_field(num_lean_medlat, num_lean_antpos, twist_ang, r_k_vals, r_preten_vals, Vars, Child, R_INTERCON_new, flag_SHOW, Boundary, file_name, true, for_paper)

            end


        else        % look how range of variables affects things
            loop_num = 1;               % used to count number of loops completed


            if flag_analisis_type == 51          % look at rope stiffness effect
                val_range = linspace(Boundary.r_k_min, Boundary.r_k_max, inc_resolution);
                var_name = "r_k";

            elseif flag_analisis_type == 52      % look at column radius effect
                val_range = linspace(Boundary.col_rad_min, Boundary.col_rad_max, inc_resolution);
                var_name = "col_rad";

            elseif flag_analisis_type == 53      % look at upper pulley offset effect
                val_range = linspace(Boundary.p_offset_min, Boundary.p_offset_max, inc_resolution);
                var_name = "upper_pull_offset";

            elseif flag_analisis_type == 54      % look at pre tension effect
                val_range = linspace(Boundary.r_preten_min, Boundary.r_preten_max, inc_resolution);
                var_name = "r_preten";

            elseif flag_analisis_type == 55      % look at harness location effect
                val_range = linspace(Boundary.harn_h_min, Boundary.harn_h_max, inc_resolution);
                var_name = "harn_hei";

            end


```

```
        for val=val_range

            % done so information is not printed for every step. Speeds up command window printing
            %if mod(loop_num, inc_resolution/num_plot_force_field) == 0 || loop_num == 1 || loop_num == inc_resolution
            fprintf('\n\n')
            track_time_elapsed(strcat("Running ", var_name, " point"), loop_num, inc_resolution)
            fprintf('\n')


            % if statement must happen again within loop too as seleced variable is constantly changing
            % change selected variable to new value
            if flag_analisis_type == 51         % look at rope stiffness effect
                Vars.r_k = val;

            elseif flag_analisis_type == 52     % look at column radius effect
                Vars.rad_col = val;

            elseif flag_analisis_type == 53     % look at upper pulley offset effect
                Vars.p_offset = val;

            elseif flag_analisis_type == 54     % look at pre tension effect
                Vars.r_preten = val;

            elseif flag_analisis_type == 55     % look at harness location effect
                Vars.h_loc = val;

            end


            % run simulation and save result
            output_data.sim_results(loop_num) = sim_device(Vars, Child, R_INTERCON, flag_SHOW, Boundary);


            if flag_run_type == 500
                % when the file should be saved
                file_name = strcat(var_name, "_", num2str(val));
            else
                file_name = "";
            end


            if flag_analisis_type == 51 || flag_analisis_type == 54
                plotForceFields(num_lean_medlat, num_lean_antpos, num_twist, Vars, Child, R_INTERCON, flag_SHOW, Boundary, file_name, true , for_paper)

            else
                plotForceFields(num_lean_medlat, num_lean_antpos, num_twist, Vars, Child, R_INTERCON, flag_SHOW, Boundary, file_name, true , for_paper)
                plotSensitivityAnalysis_field(num_lean_medlat, num_lean_antpos, twist_ang, r_k_vals, r_preten_vals, Vars, Child, R_INTERCON,
```

```
                                flag_SHOW, Boundary, file_name,true , for_paper)

                end

                loop_num=loop_num+1;        % increment loop counter
            end
        end
    end

    if flag_run_type == 500
        make_gif
    end

elseif 502 == flag_run_type
    inc_resolution = 50;        % (unit) number of points to look at per variable (
        resolution)

    % if sensitivity analysis looks at defined range
    r_k_vals = [30 100 200];          % [low medium high] rope stiffness values
    r_preten_vals = [70 110 150];     % [low medium high] pre tensioning values

    l_medlat_ang = Boundary.lean_r_max;
    l_antpos_ang = Boundary.lean_f_max;
    twist_ang = 0;

    % var_change = 1: look at rope stiffness effect
    % var_change = 2: look at column radius effect
    % var_change = 3: look at upper pulley offset effect
    % var_change = 4: look at pre tension effect
    % var_change = 5: look at harness location effect
    for var_change=[2 3 5]
        if var_change == 1
            y_lims = [0 0];       % must set this manually so its consistent across
                all plots

        elseif var_change == 2
            var_range = linspace(Boundary.col_rad_min, Boundary.col_rad_max,
                inc_resolution);
            y_lims = [-40 60];       % must set this manually so its consistent
                across all plots

        elseif var_change == 3
            var_range = linspace(Boundary.p_offset_min, Boundary.p_offset_max,
                inc_resolution);
            y_lims = [-70 90];       % must set this manually so its consistent
                across all plots

        elseif var_change == 4
            y_lims = [0 0];       % must set this manually so its consistent across
                all plots

        elseif var_change == 5
            var_range = linspace(Boundary.harn_h_min, Boundary.harn_h_max,
                inc_resolution);
            y_lims = [-30 90];       % must set this manually so its consistent
                across all plots

        end

```

```
        plotSensitivityAnalysis_9graphs(l_medlat_ang, l_antpos_ang, twist_ang,
            r_k_vals, r_preten_vals, var_change, var_range, y_lims, Vars, Child,
            R_INTERCON, flag_SHOW, Boundary, "", false, for_paper)
    end


elseif flag_run_type ~= 1
    disp("You have not set a valid flag_run_type value.")
end


end_time = toc;     % prints elapsed time from start
fprintf("----- Simulation Completed: it took %g min and %g sec -----\n\n\n\n", fix(
    end_time/60), rem(end_time, 60))
```

F.2 Simulation Code

```
%% SIMULATION FUNCTION
% takes input about the patient and rope set-up and returns net torque and
% if the torque aids child or if the child will fall
function output_data = sim_device(Vars, Child, R_INTERCON, flag_SHOW, Boundary)
    %% defining constants

    % constants about physical structure of device
    COL_half_THICK = 0.02;          % (m) Thickness of columns (asumed square)
    Thet_corners = 60;          % (deg) angle at corners of device


    % constants for ploting "3d model"
    Len_ref_star = 0.07;            % length of refference star lines
    FORCE_SCALING = 0.007;          % set scaling of forces in figure. Purely
        cosmetic


    % (m) distance from top of chair to harness location on torso
    Hei_harn = Child.t_len*(Vars.h_loc/100);
    % (m) distance from ground to upper pulleys (equivalent to global height of
        harness +- pulley offset)
    Hei_pull = Child.chair + Hei_harn + Vars.p_offset;


    % calculating distances for columns. Not ideal way but previously relied on
        length between columns
    Len_triang_side = Vars.rad_col*cosd(Thet_corners/2)*2;  % (m) distance between
        each column (sides of triangle)



    %% -----------------------------------------------------------------------
    % --- determine location of PRIMARY points (upper pulleys and harness) ---

    % left upper pulley points (assuming over origin)
    Pul_L_COL = [0 0];
    if R_INTERCON
        % used for interconnected ropes
        Pul_UL_CW = create_point((Pul_L_COL(1)-COL_half_THICK*sind(30)), (Pul_L_COL
            (2)+COL_half_THICK*cosd(30)), Hei_pull);
        Pul_UL_CCW = create_point((Pul_L_COL(1)+COL_half_THICK*cosd(60)), (Pul_L_COL
            (2)-COL_half_THICK*sind(60)), Hei_pull);
```

```

    else
        % used for separate ropes
        Pul_UL_C = create_point(Pul_L_COL(1), Pul_L_COL(2), Hei_pull);
    end


    % right upper pulley points
    Pul_R_COL = [Len_triang_side 0];
    if R_INTERCON
        % used for interconnected ropes
        Pul_UR_CW = create_point((Pul_R_COL(1)-COL_half_THICK*cosd(60)), (Pul_R_COL(2)-COL_half_THICK*sind(60)), Hei_pull);
        Pul_UR_CCW = create_point((Pul_R_COL(1)+COL_half_THICK*sind(30)), (Pul_R_COL(2)+COL_half_THICK*cosd(30)), Hei_pull);

    else
        % used for separate ropes
        Pul_UR_C = create_point(Pul_R_COL(1), Pul_R_COL(2), Hei_pull);
    end


    % front upper pulley points
    Pul_F_COL = [Len_triang_side/2 (Vars.rad_col+Vars.rad_col*sind(30))];
    if R_INTERCON
        % used for interconnected ropes
        Pul_UF_CW = create_point(Pul_F_COL(1) + COL_half_THICK, Pul_F_COL(2), Hei_pull);
        Pul_UF_CCW = create_point(Pul_F_COL(1) - COL_half_THICK, Pul_F_COL(2), Hei_pull);

    else
        % used for separate ropes
        Pul_UF_C = create_point(Pul_F_COL(1), Pul_F_COL(2), Hei_pull);
    end



    % ------------------------------------------------------------------
    % --- determine location of bottom pulleys pulleys (if they exist) ---
    if R_INTERCON
        % Location of bottom left pulleys
        Pul_BL_CW = create_point(Pul_UL_CW(1), Pul_UL_CW(2), 0);
        Pul_BL_CCW = create_point(Pul_UL_CCW(1), Pul_UL_CCW(2), 0);

        % Location of bottom right pulleys
        Pul_BR_CW = create_point(Pul_UR_CW(1), Pul_UR_CW(2), 0);
        Pul_BR_CCW = create_point(Pul_UR_CCW(1), Pul_UR_CCW(2), 0);

        % Location of bottom front pulleys
        Pul_BF_CW = create_point(Pul_UF_CW(1), Pul_UF_CW(2), 0);
        Pul_BF_CCW = create_point(Pul_UF_CCW(1), Pul_UF_CCW(2), 0);

    end



    % ------------------------------------------------------------------
    % --- determine location of starting harness points and related points ---
```

```
% harness center point (assuming 3 points for now)
Har_C_0 = create_point(Len_triang_side/2, tand(Thet_corners/2)*Len_triang_side/2, Hei_harn+Child.chair);

% define distances relating the center point to the left and right harness points
Har_x_disp = Child.t_rad*sind(60);
Har_y_disp = Child.t_rad*cosd(60);
% define the 3 points on the harness
Har_L_0 = create_point(Har_C_0(1)-Har_x_disp, Har_C_0(2)-Har_y_disp, Har_C_0(3));
Har_R_0 = create_point(Har_C_0(1)+Har_x_disp, Har_C_0(2)-Har_y_disp, Har_C_0(3));
Har_F_0 = create_point(Har_C_0(1), Har_C_0(2)+Child.t_rad, Har_C_0(3));

% patient center of mass point
Pat_CoM_0 = create_point(Har_C_0(1), Har_C_0(2), Child.CoM+Child.chair);
% patient tip of head
Pat_head_tip_0 = create_point(Har_C_0(1), Har_C_0(2), Child.chair+Child.tip_head);
% patient shoulder position
Pat_shoulders_0 = create_point(Har_C_0(1), Har_C_0(2), Child.chair+Child.t_len);

% create refference star (used for giving better understanding)
Ref_Star_0 = [Har_C_0(1)+Len_ref_star Har_C_0(2) Har_C_0(3);
              Har_C_0(1) Har_C_0(2)+Len_ref_star Har_C_0(3);
              Har_C_0(1) Har_C_0(2) Har_C_0(3)+Len_ref_star];



% center of rotaion. Assuming x and y are equal to center of harness
Poi_center_rot = create_point(Har_C_0(1), Har_C_0(2), Child.chair);



%% determine the starting runs of cables
% if the ropes are interconnected, runs between pulleys are included
if R_INTERCON
    Run_Har_Upul_0 = norm(Har_L_0 - Pul_UL_CW);      % distance from harness to upper pulley

    % get the rest of the run lengths
    Run_Upul_Bpul = norm(Pul_UL_CW - Pul_BL_CW);     % distance from upper to lower pulley
    Run_Bpul_Bpul = norm(Pul_BL_CW - Pul_BF_CCW);    % distance between lower pulleys

    Run_Cab_LR_0 = Run_Har_Upul_0*2 + Run_Upul_Bpul*2 + Run_Bpul_Bpul;  % total run length

else            % if the ropes are not interconnected
    Run_Cab_LR_0 = norm(Har_L_0 - Pul_UL_C);    % use central attachment point

end
% assuming these other runs are the same due to symetry &&& assumption of cylindrical torso
Run_Cab_RF_0 = Run_Cab_LR_0;
Run_Cab_FL_0 = Run_Cab_LR_0;
```

```
% print info about setup
if flag_SHOW.disp_text
    fprintf("Petient height is:\t%gm.\t Patient weight is:\t%gkg.\n", Child.hei, Child.wei)
    fprintf("Mediolateral lean is:\t%gdeg.\n", Vars.l_medlat)
    fprintf("Anterior-posterior lean is:\t%gdeg.\n", Vars.l_antpos)
    fprintf("Twisting is:\t%gdeg.\n", Vars.twist)
    if R_INTERCON
        fprintf("3 ropes are interconnected.\n\n")
    else
        fprintf("3 ropes are separate.\n\n")
    end
end



%% starting rope length properties

% F = Ku        &&& assuming perfectly elastic ropes based on Hooke's law
% (N) initial extended length of cables
u_LR_0 = Vars.r_preten/Vars.r_k;
u_RF_0 = Vars.r_preten/Vars.r_k;
u_FL_0 = Vars.r_preten/Vars.r_k;


% ---------- Determine unstretched length of the cables ----------
% If we know the original length of the cable going through the runs, then it is easier to
% compute the force in cables whilst taking into account pre-tensioning of the cables. This
% pre-tensioning inherently lenghtenss the ropes hence why the unstretched lengths are shorter
% than the runs they go through (and subsiquently their stretched lengths).
Len_Cab_LR_Unstr = Run_Cab_LR_0-u_LR_0;
Len_Cab_RF_Unstr = Run_Cab_RF_0-u_RF_0;
Len_Cab_FL_Unstr = Run_Cab_FL_0-u_FL_0;


% calculate percentage rope is elongated
r_stretch_perc_0 = (u_LR_0/Len_Cab_LR_Unstr)*100;


% check to see unstretched lengths are not less than zero. If they are, means the rope needs to
% be of a negative length as u is longer than the run which is not a possiblity
if Len_Cab_LR_Unstr <= 0
    if flag_SHOW.disp_text
        fprintf(2, "There is an issue with unstretched rope lengths being negative.\n" + ...
         "Left-right cable = %gm.\nRight-front cable = %gm.\nFront-left cable = %gm.\n\t---flagging anomaly---\n\n", ...
         Len_Cab_LR_Unstr, Len_Cab_RF_Unstr, Len_Cab_FL_Unstr)
    end
    Flag_rope_unstretch_good = false;
else
    Flag_rope_unstretch_good = true;
end
```

```matlab


    % check to see how the stretched length of the cable compares to the original. if it is over the
    % specified ammount indicated by the rope specs, then the rope is not suitable
    if r_stretch_perc_0 >= Boundary.r_stretch_percent_max
        if flag_SHOW.disp_text
            fprintf(2, "There is a very high rope stretch proportion.\n" + ...
                "Current rope stretch proportion = %g.\nMaximum allowed rope stretch percent = %g.\n\nflagging anomaly\n\n", ...
                u_LR_0/Len_Cab_LR_Unstr, Boundary.r_stretch_percent_max)
        end
        Flag_rope_stretch_0_good = false;
    else
        Flag_rope_stretch_0_good = true;
    end


    %% Patient new position

    % use rotation matrix to find new position of each point
    har_l_lean = rotate_vector(Har_L_0, Poi_center_rot, Vars.l_antpos, Vars.l_medlat, Vars.twist);
    har_r_lean = rotate_vector(Har_R_0, Poi_center_rot, Vars.l_antpos, Vars.l_medlat, Vars.twist);
    har_f_lean = rotate_vector(Har_F_0, Poi_center_rot, Vars.l_antpos, Vars.l_medlat, Vars.twist);
    har_c_lean = rotate_vector(Har_C_0, Poi_center_rot, Vars.l_antpos, Vars.l_medlat, Vars.twist);
    Pat_CoM_lean = rotate_vector(Pat_CoM_0, Poi_center_rot, Vars.l_antpos, Vars.l_medlat, Vars.twist);
    Ref_Star_lean = rotate_vector(Ref_Star_0, Poi_center_rot, Vars.l_antpos, Vars.l_medlat, Vars.twist);
    Pat_shoulders = rotate_vector(Pat_shoulders_0, Poi_center_rot, Vars.l_antpos, Vars.l_medlat, Vars.twist);
    Pat_head_tip = rotate_vector(Pat_head_tip_0, Poi_center_rot, Vars.l_antpos, Vars.l_medlat, Vars.twist);


    % if lengths are the same then we know the axis of rotation is same
    % distance from center of circle
    len_torso_0 = norm(change_origin(Poi_center_rot, Har_C_0));
    len_torso_end = norm(change_origin(Poi_center_rot, har_c_lean));


    % Used to check to make sure things are still making sence
    % have to round as matlab may not see them as 100% perfectly the same
    % 5 decimal places should be well good enough
    if round(len_torso_0, 5) == round(len_torso_end, 5)
        Flag_trunk_len_good = true;
    else
        if flag_SHOW.disp_text
            disp("Something is wrong, lengths do not match. Flagging issue")
            len_torso_0
            len_torso_end
        end
        Flag_trunk_len_good = false;
    end

```

```
    %% calculate forces in rope

    % find the length of each rope at new position
    % when 3 interconnected ropes are used
    if R_INTERCON
        % find lengths of the rope runs
        run_har_upul_LCW = norm(change_origin(Pul_UL_CW, har_l_lean));
        run_har_upul_LCCW = norm(change_origin(Pul_UL_CCW, har_l_lean));
        run_har_upul_RCW = norm(change_origin(Pul_UR_CW, har_r_lean));
        run_har_upul_RCCW = norm(change_origin(Pul_UR_CCW, har_r_lean));
        run_har_upul_FCW = norm(change_origin(Pul_UF_CW, har_f_lean));
        run_har_upul_FCCW = norm(change_origin(Pul_UF_CCW, har_f_lean));

        len_cab_LR_end = Run_Bpul_Bpul+2*Run_Upul_Bpul + run_har_upul_LCCW + run_har_upul_RCW;  % left to right cable
        len_cab_RF_end = Run_Bpul_Bpul+2*Run_Upul_Bpul + run_har_upul_RCCW + run_har_upul_FCW;  % right to front cable
        len_cab_FL_end = Run_Bpul_Bpul+2*Run_Upul_Bpul + run_har_upul_FCCW + run_har_upul_LCW;  % front to left cable


    % when 3 separate ropes are used
    else
        len_cab_LR_end = norm(change_origin(Pul_UL_C, har_l_lean));  % just for the left cable
        len_cab_RF_end = norm(change_origin(Pul_UR_C, har_r_lean));  % just for the right cable
        len_cab_FL_end = norm(change_origin(Pul_UF_C, har_f_lean));  % just for the front cable

    end


    % determine stretch length of cables
    u_LR_end = len_cab_LR_end - Len_Cab_LR_Unstr;
    u_RF_end = len_cab_RF_end - Len_Cab_RF_Unstr;
    u_FL_end = len_cab_FL_end - Len_Cab_FL_Unstr;


    % check to see if stretch is negative. If so set to 0 and inform of oddity
    % <=0 value indicates slack in rope
    if u_LR_end <= 0
        u_LR_end = 0;

        if flag_SHOW.disp_text
            fprintf(2,"There is no tension in Left-Right rope.\n")
        end
        Flag_no_slack_lr = false;
    else
        Flag_no_slack_lr = true;
    end
    if u_RF_end <= 0
        u_RF_end = 0;

        if flag_SHOW.disp_text
            fprintf(2,"There is no tension in Right-Front rope.\n")
        end
```

```
        Flag_no_slack_rf = false;
    else
        Flag_no_slack_rf = true;
    end
    if u_FL_end <= 0
        u_FL_end = 0;

        if flag_SHOW.disp_text
            fprintf(2,"There is no tension in Front-Left rope.\n")
        end
        Flag_no_slack_fl = false;
    else
        Flag_no_slack_fl = true;
    end


    % get percentage ropes are stretched
    r_stretch_perc_lr = (u_LR_end/Len_Cab_LR_Unstr)*100;
    r_stretch_perc_rf = (u_RF_end/Len_Cab_RF_Unstr)*100;
    r_stretch_perc_fl = (u_FL_end/Len_Cab_FL_Unstr)*100;

    r_stretch_perc_end_max = max([r_stretch_perc_lr r_stretch_perc_rf r_stretch_perc_fl]);


    % check to see how rope stretch compares to unstretched rope. If rope stretch is >= given
    % stretch proportion of unstretched length, flag this as an issue as this is a very high stretch
    % proportion. For vague estimate, found this:
    % https://www.ropesdirect.co.uk/blog/everything-you-need-to-know-about-bungee-cord/
    if r_stretch_perc_lr >= Boundary.r_stretch_percent_max
        if flag_SHOW.disp_text
            fprintf(2, "There is a very high rope stretch proportion in Left-right rope.\n" + ...
                "Current rope stretch proportion = %g.\nMaximum allowed rope stretch percent = %g.\n\nflagging anomaly\n\n", ...
                u_LR_0/Len_Cab_LR_Unstr, Boundary.r_stretch_percent_max)
        end
        Flag_rope_stretch_lr_1_good = false;
    else
        Flag_rope_stretch_lr_1_good = true;
    end

    if r_stretch_perc_rf >= Boundary.r_stretch_percent_max
        if flag_SHOW.disp_text
            fprintf(2, "There is a very high rope stretch proportion in Right-front rope.\n" + ...
                "Current rope stretch proportion = %g.\nMaximum allowed rope stretch percent = %g.\n\nflagging anomaly\n\n", ...
                u_LR_0/Len_Cab_LR_Unstr, Boundary.r_stretch_percent_max)
        end
        Flag_rope_stretch_rf_1_good = false;
    else
        Flag_rope_stretch_rf_1_good = true;
    end

    if r_stretch_perc_fl >= Boundary.r_stretch_percent_max
```

```
        if flag_SHOW.disp_text
            fprintf(2, "There is a very high rope stretch proportion in Front-left rope.\n" + ...
                "Current rope stretch proportion = %g.\nMaximum allowed rope stretch percent = %g.\n\nflagging anomaly\n\n", ...
                u_LR_0/Len_Cab_LR_Unstr, Boundary.r_stretch_percent_max)
        end
        Flag_rope_stretch_fl_1_good = false;
    else
        Flag_rope_stretch_fl_1_good = true;
    end



    % F = Ku
    % compute forces in each cable
    fmag_LR_end = Vars.r_k*u_LR_end;
    fmag_RF_end = Vars.r_k*u_RF_end;
    fmag_FL_end = Vars.r_k*u_FL_end;

    % find maximum tension in any rope
    fmag_rope_max_end = max(abs([fmag_LR_end fmag_RF_end fmag_FL_end]));



    % display force information within the ropes themselves
    if flag_SHOW.disp_text
        fprintf("=== At the given location: ===\n")
        fprintf("Tension force in left-right cable is:\t%gN.\n", fmag_LR_end)
        fprintf("Tension force in right-front cable is:\t%gN.\n", fmag_RF_end)
        fprintf("Tension force in front-left cable is:\t%gN.\n\n", fmag_FL_end)
    end



    %% calculate forces acting on each harness point
    % alter calculation based on if ropes are interconnected or not
    if R_INTERCON
        fvec_LCW_end = get_Fvec_vec(fmag_FL_end, (Pul_UL_CW-har_l_lean));
        fvec_LCCW_end = get_Fvec_vec(fmag_LR_end, (Pul_UL_CCW-har_l_lean));
        fvec_RCW_end = get_Fvec_vec(fmag_LR_end, (Pul_UR_CW-har_r_lean));
        fvec_RCCW_end = get_Fvec_vec(fmag_RF_end, (Pul_UR_CCW-har_r_lean));
        fvec_FCW_end = get_Fvec_vec(fmag_RF_end, (Pul_UF_CW-har_f_lean));
        fvec_FCCW_end = get_Fvec_vec(fmag_FL_end, (Pul_UF_CCW-har_f_lean));

        fvec_l_end = fvec_LCW_end + fvec_LCCW_end;
        fvec_r_end = fvec_RCW_end + fvec_RCCW_end;
        fvec_f_end = fvec_FCW_end + fvec_FCCW_end;

    else
        % &&& using clocwise harness points for now. Should use a central point on the column
        fvec_l_end = get_Fvec_vec(fmag_LR_end, (Pul_UL_C-har_l_lean));
        fvec_r_end = get_Fvec_vec(fmag_RF_end, (Pul_UR_C-har_r_lean));
        fvec_f_end = get_Fvec_vec(fmag_FL_end, (Pul_UF_C-har_f_lean));      % &&& we have a problem

    end

```

```
    % display force information within the ropes themselves
    if flag_SHOW.disp_text
        fprintf("The magnitude of the force acting on the left harness point is: \t%gN.\n", norm(fvec_l_end))
        fprintf("The magnitude of the force acting on the right harness point is:\t%gN.\n", norm(fvec_r_end))
        fprintf("The magnitude of the force acting on the front harness point is:\t%gN.\n\n", norm(fvec_f_end))
    end


    %% determine if device could right child in event they cannot provide any support
    % get gravity force vector
    F_grav_mag = Child.wei*Child.t_wei_prop*9.81;   % magnitude of gravity
    F_grav_vec = [0 0 -1]*F_grav_mag;               % creat F_g vector


    % lever arms for each harness point and gravity
    lev_arm_vec_c = change_origin(Poi_center_rot, har_c_lean);
    lev_arm_vec_l = change_origin(Poi_center_rot, har_l_lean);
    lev_arm_vec_r = change_origin(Poi_center_rot, har_r_lean);
    lev_arm_vec_f = change_origin(Poi_center_rot, har_f_lean);
    lev_arm_vec_g = change_origin(Poi_center_rot, Pat_CoM_lean);

    lev_arm_harn_mag = norm(lev_arm_vec_c);
    lev_arm_grav_mag = norm(lev_arm_vec_g);


    torso_direciton = change_origin(Poi_center_rot, Pat_CoM_lean);    % vector that is perpendicular to plane of harness


    % find component that is parallel to the torso direciton and does not help move child
    % https://www.youtube.com/watch?v=fqPiDICPkj8
    fvec_comp_l_para = project_v1_on_v2(fvec_l_end,torso_direciton);
    fvec_comp_r_para = project_v1_on_v2(fvec_r_end,torso_direciton);
    fvec_comp_f_para = project_v1_on_v2(fvec_f_end,torso_direciton);
    fvec_comp_g_para = project_v1_on_v2(F_grav_vec,torso_direciton);
    % get magnitude for each vector
    mag_Fl_comp_para = norm(fvec_comp_l_para);
    mag_Fr_comp_para = norm(fvec_comp_r_para);
    mag_Ff_comp_para = norm(fvec_comp_f_para);


    % find components perpendicular to torso direction (parallel to harness plane and do move child)
    fvec_comp_l_move = fvec_l_end - fvec_comp_l_para;
    fvec_comp_r_move = fvec_r_end - fvec_comp_r_para;
    fvec_comp_f_move = fvec_f_end - fvec_comp_f_para;
    fvec_comp_g_move = F_grav_vec - fvec_comp_g_para;
    % get magnitude for each vector
    mag_Fl_comp_move = norm(fvec_comp_l_move);
    mag_Fr_comp_move = norm(fvec_comp_r_move);
    mag_Ff_comp_move = norm(fvec_comp_f_move);
    mag_Fg_comp_move = norm(fvec_comp_g_move);
```

```
% get net forces related to the harness (total, parasitic, movement generating)
fnet_harn = fvec_l_end + fvec_r_end + fvec_f_end;
fnet_harn_para = fvec_comp_l_para + fvec_comp_r_para + fvec_comp_f_para;
fnet_harn_move = fvec_comp_l_move + fvec_comp_r_move + fvec_comp_f_move;


% get torques
tnet_vec = fnet_harn_move*lev_arm_harn_mag + fvec_comp_g_move*lev_arm_grav_mag;
tnet_mag = norm(tnet_vec);



%% know, mathematically, if torque will correct or child will fall over

% --- finding vector torque direction to center ---
% https://www.youtube.com/watch?v=imBrK9QB5AA
% find angle from z axis
lean_z_height = har_c_lean(3)-Poi_center_rot(3);    % find vertical height component (origin = center of rotation)

lean_thet = acosd(lean_z_height/Hei_harn);          % Hei_harn = hypotenuse

% find point of intersection with z axis
%   switching from completely horizontal motion from lean point to z axis
%   to an angled direction to z axis 90 deg to lean vector
lean_len_orthog_z_intersect = Hei_harn/cosd(lean_thet);
lean_z_intersect = create_point(0, 0, lean_len_orthog_z_intersect) + Poi_center_rot;

% direction of torque that would most directly correct the patient from the currect position
torq_correcting_direc_vect = change_origin(har_c_lean, lean_z_intersect);


% know, mathematically, if torque will correct or child will fall over
% get the unit vector of the torque and righting direction vectors
torq_net_direc = tnet_vec/norm(tnet_vec);
direc_to_correct = torq_correcting_direc_vect/norm(torq_correcting_direc_vect);

% main way it was done but ang_tdirec_dif now used since result is more easily understandable
direction_torque = torq_net_direc/direc_to_correct;

% find angle (deg) difference between the ideal torque direction and the real torque
ang_tdirec_dif_pre = dot(tnet_vec,torq_correcting_direc_vect)/dot(norm(tnet_vec),norm(torq_correcting_direc_vect));
ang_tdirec_dif = acosd(round(ang_tdirec_dif_pre, 10));      % rounding else there may be an imaginary output



% &&& maybe need something to check the value, if the directions don't match well then send a
% warning
if flag_SHOW.disp_text
    if direction_torque > 0
```

```
            fprintf("Net torque acting on the child WOULD automatically right them from this position.\n\n");
        else
            fprintf(2, "Net torque acting on the child will cause them to FALL from this position.\n" + ...
                "Further changes are needed to make this safe\n\n");
        end
    end

    %% find largest parasitic force
    % use value of largest index to save that value and retain the sign
    f_mag_para_max = max(abs([mag_Fl_comp_para mag_Fr_comp_para mag_Ff_comp_para]));



    %% find moments induced on the child by the 3 harness points

    % find the directions of the tangents at each point
    tan_r = cross(torso_direciton, change_origin(har_c_lean, har_l_lean));
    tan_l = cross(torso_direciton, change_origin(har_c_lean, har_r_lean));
    tan_f = cross(torso_direciton, change_origin(har_c_lean, har_f_lean));


    % find the component of the moving force component that is parallel to the tangent
    fvec_comp_l_moment = project_v1_on_v2(fvec_comp_l_move, tan_r);
    fvec_comp_r_moment = project_v1_on_v2(fvec_comp_r_move, tan_l);
    fvec_comp_f_moment = project_v1_on_v2(fvec_comp_f_move, tan_f);


    % net twisting moment
    torq_net_twist = Child.t_rad*(fvec_comp_l_moment + fvec_comp_r_moment + fvec_comp_f_moment);
    torq_net_twist_mag = norm(torq_net_twist);


    % Used to check to make sure things are still making sence
    % find magnitudes of everything
    mag_Fl_comp_mom = norm(fvec_comp_l_moment);
    mag_Fr_comp_mom = norm(fvec_comp_r_moment);
    mag_Ff_comp_mom = norm(fvec_comp_f_moment);

    torq_net_twist_mag2 = mag_Fl_comp_mom + mag_Fr_comp_mom + mag_Ff_comp_mom;



    % have to round as matlab may not see them as 100% perfectly the same
    % 5 decimal places should be well good enough
    if flag_SHOW.disp_text
        if round(mag_Fl_comp_move, 5) >= round(mag_Fl_comp_mom, 5)
            fprintf("Moments for left seem good. Total mag is:\t%gN \t and torque component mag is:\t\t%gN.\n", mag_Fl_comp_move, mag_Fl_comp_mom);
        else
            disp("Something is wrong, forces for left seem a bit strange")
            mag_Fl_comp_move
            mag_Fl_comp_mom
        end
        if round(mag_Fr_comp_move, 5) >= round(mag_Fr_comp_mom, 5)
            fprintf("Moments for right seem good. Total mag is:\t%gN \t and torque
```

```
                component mag is:\t\t%gN.\n", mag_Fr_comp_move, mag_Fr_comp_mom);
        else
            disp("Something is wrong, forces for right seem a bit strange")
            mag_Fr_comp_move
            mag_Fr_comp_mom
        end
        if round(mag_Ff_comp_move, 5) >= round(mag_Ff_comp_mom, 5)
            fprintf("Moments for front seem good. Total mag is:\t%gN \t and torque component mag is:\t\t%gN.\n", mag_Ff_comp_move, mag_Ff_comp_mom);
        else
            disp("Something is wrong, forces for front seem a bit strange")
            mag_Ff_comp_move
            mag_Ff_comp_mom
        end

        fprintf("Net torque magnitude is:\t%gNm\n\n", tnet_mag);
    end



    if flag_SHOW.disp_text
        fprintf("The parasitic force at the left harness point has a magnitude of: \t%gN.\n", mag_Fl_comp_para);
        fprintf("The parasitic force at the right harness point has a magnitude of: \t%gN.\n", mag_Fr_comp_para);
        fprintf("The parasitic force at the front harness point has a magnitude of: \t%gN.\n", mag_Ff_comp_para);


        if mag_Fl_comp_para > mag_Fr_comp_para && mag_Fl_comp_para > mag_Ff_comp_para
            biggest_para_name = "left";
            biggest_para_val = mag_Fl_comp_para;

        elseif mag_Fr_comp_para > mag_Fl_comp_para && mag_Fr_comp_para > mag_Ff_comp_para
            biggest_para_name = "right";
            biggest_para_val = mag_Fr_comp_para;

        elseif mag_Ff_comp_para > mag_Fr_comp_para && mag_Ff_comp_para > mag_Fl_comp_para
            biggest_para_name = "front";
            biggest_para_val = mag_Ff_comp_para;

        elseif mag_Fl_comp_para == mag_Fr_comp_para && mag_Fl_comp_para == mag_Ff_comp_para && mag_Fr_comp_para == mag_Ff_comp_para
            biggest_para_name = "all";
            biggest_para_val = mag_Fl_comp_para;

        elseif mag_Fl_comp_para == mag_Fr_comp_para
            biggest_para_name = "left and right";
            biggest_para_val = mag_Fl_comp_para;

        elseif mag_Fl_comp_para == mag_Ff_comp_para
            biggest_para_name = "left and front";
            biggest_para_val = mag_Fl_comp_para;

        elseif mag_Fr_comp_para == mag_Ff_comp_para
            biggest_para_name = "right and front";
```

```
            biggest_para_val = mag_Fr_comp_para;

        else
            biggest_para_name = "none";
            biggest_para_val = 0;

        end


        fprintf("The highest parasitic force was found at the %s harness point. Its
            magnitude was: \t%gN.\n", biggest_para_name, biggest_para_val);

    end


    %% plot "3D model" of desired
    if flag_SHOW.disp_3d_model
        % define custom colors
        col_green = [0 0.8 0];
        grayColor = [.7 .7 .7];
        rope_color = [.3 .3 .3];

        % determine lbl offset
        lbl_x_offset = 5;
        lbl_y_offset = 5;

        % define size of points
        p_size_har = 25;
        p_size_pull = 10;
        p_size_other = 7;
        p_size_extra = 3;

        % define style of points
        p_style_har = '.';
        p_style_pull = 'o';
        p_style_other = '*';
        p_style_extra = '*';


        % set up figure nicely
        fig_labs = ["X (m)" "Y (m)" "Z (m)"];
        fig_lims = [-0.1 3; -0.1 2.5; 0 0];
        setup_fig("fig", [0 0 0], [1 1 1], "ropes", 0, R_INTERCON, [0 -90], fig_labs
            , fig_lims, true, true, true);
        % [-52.924265327415206 35.212596336003756]
        % [-54.743532992464878 29.922107967541887]
        % [180 -90]
        % [-90 0]
        % [0 110]

        if flag_SHOW.radius_circle
            % a cricle indicating radius from center
            plotCircle3D([Har_C_0(1) Har_C_0(2) 0], [0 0 1], Vars.rad_col, 'k:')

            plot_line([Har_C_0(1) Har_C_0(2) 0], [Pul_UR_C(1) Pul_UR_C(2) 0], ':', '
                k')
            plot_point([abs(Har_C_0(1)/2-Pul_UR_C(1)) abs(Har_C_0(2)/2-Pul_UR_C(2))
                0], "", 5, 'k', strcat('Col Rad=', num2str(Vars.rad_col), "m"), '
                left')
```

```
        end


        % plot pulley locations
        if R_INTERCON
            if flag_SHOW.main_lables
                % when the lables should be shown
                lbl_pull(1) = "UL_{CW}";
                lbl_pull(2) = "UL_{CCW}";
                lbl_pull(3) = "UR_{CW}";
                lbl_pull(4) = "UR_{CCW}";
                lbl_pull(5) = "UF_{CW}";
                lbl_pull(6) = "UF_{CCW}";
                lbl_pull(7) = "BL_{CW}";
                lbl_pull(8) = "BL_{CCW}";
                lbl_pull(9) = "BR_{CW}";
                lbl_pull(10) = "BR_{CCW}";
                lbl_pull(11) = "BF_{CW}";
                lbl_pull(12) = "BF_{CCW}";
            else
                % when the lables should not be shown
                for i=1:12
                    lbl_pull(i) = "";
                end
            end
            plot_point(Pul_UL_CW, p_style_pull, p_size_pull, 'b', lbl_pull(1), '
                right')
            plot_point(Pul_UL_CCW, p_style_pull, p_size_pull, 'b', lbl_pull(2), '
                left')
            plot_point(Pul_UR_CW, p_style_pull, p_size_pull, 'r', lbl_pull(3), '
                right')
            plot_point(Pul_UR_CCW, p_style_pull, p_size_pull, 'r', lbl_pull(4), '
                left')
            plot_point(Pul_UF_CW, p_style_pull, p_size_pull, col_green, lbl_pull(5),
                 'left')
            plot_point(Pul_UF_CCW, p_style_pull, p_size_pull, col_green, lbl_pull(6)
                , 'right')
            plot_point(Pul_BL_CW, p_style_pull, p_size_pull, 'b', lbl_pull(7), '
                right')
            plot_point(Pul_BL_CCW, p_style_pull, p_size_pull, 'b', lbl_pull(8), '
                left')
            plot_point(Pul_BR_CW, p_style_pull, p_size_pull, 'r', lbl_pull(9), '
                right')
            plot_point(Pul_BR_CCW, p_style_pull, p_size_pull, 'r', lbl_pull(10), '
                left')
            plot_point(Pul_BF_CW, p_style_pull, p_size_pull, col_green, lbl_pull(11)
                , 'left')
            plot_point(Pul_BF_CCW, p_style_pull, p_size_pull, col_green, lbl_pull
                (12), 'right')

        else
            if flag_SHOW.main_lables
                % when the lables should be shown
                lbl_pull(1) = "UL";
                lbl_pull(2) = "UR";
                lbl_pull(3) = "UF";
            else
                % when the lables should not be shown
```

```
                for i=1:3
                    lbl_pull(i) = "";
                end
            end
            plot_point(Pul_UL_C, p_style_pull, p_size_pull, 'b', lbl_pull(1), 'right')
            plot_point(Pul_UR_C, p_style_pull, p_size_pull, 'r', lbl_pull(2), 'right')
            plot_point(Pul_UF_C, p_style_pull, p_size_pull, col_green, lbl_pull(3), 'left')
        end



        % plot center of ration, CoM
        if flag_SHOW.main_lables
            % when the lables should be shown
            lbl_other(1) = "Cent\_rot";
            lbl_other(2) = "CoM";
        else
            % when the lables should not be shown
            lbl_other(1) = "";
            lbl_other(2) = "";
        end
        plot_point(Poi_center_rot, p_style_other, p_size_other, 'm', lbl_other(1), 'left')
        plot_point(Pat_CoM_lean, p_style_other, p_size_other, 'm', lbl_other(2), 'right')


        % plot top of head and shoulders
        if flag_SHOW.extra_lables
            % when the lables should be shown
            lbl_extr(1) = "Head top";
            lbl_extr(2) = "Shoulder top";
        else
            % when the lables should not be shown
            lbl_extr(1) = "";
            lbl_extr(2) = "";
        end
        plot_point(Pat_head_tip, p_style_extra, p_size_extra, 'm', lbl_extr(1), 'left')
        plot_point(Pat_shoulders, p_style_extra, p_size_extra, 'm', lbl_extr(2), 'left')



        if flag_SHOW.start_location && ~(Vars.l_medlat == 0 && Vars.l_antpos == 0 && Vars.twist == 0)
            % plot harness start location (these 3 are ploted at same level)
            if flag_SHOW.main_lables
                % when the lables should be shown
                lbl_har_0(1) = "H_{l-0}";
                lbl_har_0(2) = "H_{r-0}";
                lbl_har_0(3) = "H_{f-0}";
            else
                % when the lables should not be shown
                for i=1:3
                    lbl_har_0(i) = "";
```

```
                end
            end
            plot_point(Har_L_0, p_style_har, p_size_har, 'b', lbl_har_0(1), 'left')
            plot_point(Har_R_0, p_style_har, p_size_har, 'r', lbl_har_0(2), 'left')
            plot_point(Har_F_0, p_style_har, p_size_har, col_green, lbl_har_0(3), 'left')

            % plot starting harness poistion
            plotCircle3D(Har_C_0,cross(Har_R_0-Har_L_0, Har_F_0-Har_L_0), Child.t_rad, 'm:')

            % starting torso
            plot_line(Poi_center_rot, Pat_head_tip_0, '--', 'c')
        end


        % plot harness end location
        if flag_SHOW.main_lables
            % when the lables should be shown
            lbl_har_l(1) = "H_l";
            lbl_har_l(2) = "H_r";
            lbl_har_l(3) = "H_f";
        else
            % when the lables should not be shown
            for i=1:3
                lbl_har_l(i) = "";
            end
        end
        plot_point(har_l_lean, p_style_har, p_size_har, 'b', lbl_har_l(1), 'left')
        plot_point(har_r_lean, p_style_har, p_size_har, 'r', lbl_har_l(2), 'left')
        plot_point(har_f_lean, p_style_har, p_size_har, col_green, lbl_har_l(3), 'left')

        % make circle indicating harness
        plotCircle3D(har_c_lean,cross(har_r_lean-har_l_lean, har_f_lean-har_l_lean), Child.t_rad,'m-')



        % plot other cable connections if ropes are interconnected
        if flag_SHOW.extra_lables && R_INTERCON
            % when the lables for interconnected ropes should be shown
            lbl_rope(1) = "Rope_{L-R}";
            lbl_rope(2) = "Rope_{R-F}";
            lbl_rope(3) = "Rope_{F-L}";

        elseif flag_SHOW.extra_lables && ~R_INTERCON
            % when the lables for separate ropes should be shown
            lbl_rope(1) = "Rope_{L}";
            lbl_rope(2) = "Rope_{R}";
            lbl_rope(3) = "Rope_{F}";

        else
            % when the lables should not be shown
            lbl_rope(1) = "";
            lbl_rope(2) = "";
            lbl_rope(3) = "";
        end
        if R_INTERCON
```

```
            plot_line(Pul_UL_CW, Pul_BL_CW, '-', rope_color)
            plot_line(Pul_UL_CCW, Pul_BL_CCW, '-', rope_color)
            plot_line(Pul_UR_CW, Pul_BR_CW, '-', rope_color)
            plot_line(Pul_UR_CCW, Pul_BR_CCW, '-', rope_color)
            plot_line(Pul_UF_CW, Pul_BF_CW, '-', rope_color)
            plot_line(Pul_UF_CCW, Pul_BF_CCW, '-', rope_color)
            plot_line(Pul_BL_CW, Pul_BF_CCW, '-', rope_color)
            plot_line(Pul_BF_CW, Pul_BR_CCW, '-', rope_color)
            plot_line(Pul_BR_CW, Pul_BL_CCW, '-', rope_color)

            % label ropes
            plot_point(Pul_BR_CW/2, "", 5, rope_color, lbl_rope(1), 'left')
            plot_point([1.35 0.8 0], "", 5, rope_color, lbl_rope(2), 'left')
            plot_point(Pul_BF_CCW/2, "", 5, rope_color, lbl_rope(3), 'left')

        else
            plot_line(Pul_UL_C, [Pul_UL_C(1) Pul_UL_C(2) 0], '-', 'm')
            plot_line(Pul_UR_C, [Pul_UR_C(1) Pul_UR_C(2) 0], '-', 'm')
            plot_line(Pul_UF_C, [Pul_UF_C(1) Pul_UF_C(2) 0], '-', 'm')

            % label ropes
            plot_point([Pul_UL_C(1)+0.2 Pul_UL_C(2)+0.2 Pul_UL_C(3)-0.1], "", 5,
                rope_color, lbl_rope(1), 'left')
            plot_point([Pul_UR_C(1)-0.2 Pul_UR_C(2)+0.2 Pul_UR_C(3)-0.1], "", 5,
                rope_color, lbl_rope(2), 'left')
            plot_point([Pul_UF_C(1) Pul_UF_C(2)-0.2 Pul_UF_C(3)-0.1], "", 5,
                rope_color, lbl_rope(3), 'left')
        end


        % plot line indicating seat and participant
        plot_line(Poi_center_rot, [Poi_center_rot(1) Poi_center_rot(2) 0], '-', 'm')
        plot_line(Poi_center_rot, Pat_head_tip, '-', 'c')


        if flag_SHOW.t_net_vec
            % plot correcting torque
            % change name based on if the magniutde should be included or not
            if flag_SHOW.vector_magnitude
                lbl_T = strcat('T_{net}=', num2str(round(norm(tnet_vec), 3)));
            else
                lbl_T = 'T_{net}';
            end

            plot_force_vec(har_c_lean, tnet_vec, '-', 'm', lbl_T, 'left',
                FORCE_SCALING, 0)
        end



        if flag_SHOW.start_location && ~(Vars.l_medlat == 0 && Vars.l_antpos == 0 &&
            Vars.twist == 0)
            % plot ropes going to starting harness points
            if R_INTERCON
                plot_line(Pul_UL_CW, Har_L_0, '--', rope_color)
                plot_line(Pul_UL_CCW, Har_L_0, '--', rope_color)
                plot_line(Pul_UR_CW, Har_R_0, '--', rope_color)
                plot_line(Pul_UR_CCW, Har_R_0, '--', rope_color)
                plot_line(Pul_UF_CW, Har_F_0, '--', rope_color)
```

```
                plot_line(Pul_UF_CCW, Har_F_0, '--', rope_color)
            else
                plot_line(Pul_UL_C, Har_L_0, '--', rope_color)
                plot_line(Pul_UR_C, Har_R_0, '--', rope_color)
                plot_line(Pul_UF_C, Har_F_0, '--', rope_color)
            end


            if flag_SHOW.ref_stars
                % plot refference star at starting point
                plot_line(Har_C_0, Ref_Star_0(1,:), ":", [0.6350 0.0780 0.1840])
                plot_line(Har_C_0, Ref_Star_0(2,:), ":", [0.4660 0.6740 0.1880])
                plot_line(Har_C_0, Ref_Star_0(3,:), ":", [0 0.4470 0.7410])
            end
        end


        % plot ropes going to leaning harness points
        if R_INTERCON
            plot_line(Pul_UL_CW, har_l_lean, '-', rope_color)
            plot_line(Pul_UL_CCW, har_l_lean, '-', rope_color)
            plot_line(Pul_UR_CW, har_r_lean, '-', rope_color)
            plot_line(Pul_UR_CCW, har_r_lean, '-', rope_color)
            plot_line(Pul_UF_CW, har_f_lean, '-', rope_color)
            plot_line(Pul_UF_CCW, har_f_lean, '-', rope_color)

        else
            plot_line(Pul_UL_C, har_l_lean, '-', rope_color)
            plot_line(Pul_UR_C, har_r_lean, '-', rope_color)
            plot_line(Pul_UF_C, har_f_lean, '-', rope_color)
        end


        if flag_SHOW.ref_stars
            % plot refference star at lean point
            plot_line(har_c_lean, Ref_Star_lean(1,:), "-", [0.6350 0.0780 0.1840])
            plot_line(har_c_lean, Ref_Star_lean(2,:), "-", [0.4660 0.6740 0.1880])
            plot_line(har_c_lean, Ref_Star_lean(3,:), "-", [0 0.4470 0.7410])

            % plot refference star at center of rotation
            plot_line(Poi_center_rot, Poi_center_rot+[Len_ref_star 0 0], "-",
                [0.6350 0.0780 0.1840])
            plot_line(Poi_center_rot, Poi_center_rot+[0 Len_ref_star 0], "-",
                [0.4660 0.6740 0.1880])
            plot_line(Poi_center_rot, Poi_center_rot+[0 0 Len_ref_star], "-", [0
                0.4470 0.7410])
        end


        % plot net forces at harness and gravity points
        if flag_SHOW.rope_force

            % change name based on if the magniutde should be included or not
            if flag_SHOW.vector_magnitude
                lbl_F_l = strcat('F_l = ', num2str(norm(fvec_l_end)));
                lbl_F_r = strcat('F_r = ', num2str(norm(fvec_r_end)));
                lbl_F_f = strcat('F_f = ', num2str(norm(fvec_f_end)));
                lbl_F_g = strcat('F_{g (scaled)} = ', num2str(norm(F_grav_vec)));

```

```
            else
                lbl_F_l = 'F_l';
                lbl_F_r = 'F_r';
                lbl_F_f = 'F_f';
                lbl_F_g = 'F_{g (scaled)}';

            end

            if flag_SHOW.net_force_at_center
                % plot forces acting on harness at the refference star
                plot_force_vec(har_c_lean, fvec_l_end, '-', 'b', lbl_F_l, 'left', FORCE_SCALING, 0)
                plot_force_vec(har_c_lean, fvec_r_end, '-', 'r', lbl_F_r, 'left', FORCE_SCALING, 0)
                plot_force_vec(har_c_lean, fvec_f_end, '-', col_green, lbl_F_f, 'left', FORCE_SCALING, 0)

            else
                % plot net forces acting on harness at the harness points
                plot_force_vec(har_l_lean, fvec_l_end, '-', 'b', lbl_F_l, 'left', FORCE_SCALING, 0)
                plot_force_vec(har_r_lean, fvec_r_end, '-', 'r', lbl_F_r, 'left', FORCE_SCALING, 0)
                plot_force_vec(har_f_lean, fvec_f_end, '-', col_green, lbl_F_f, 'left', FORCE_SCALING, 0)

                % lever arms for each harness point
                plot_line((lev_arm_vec_l + Poi_center_rot), Poi_center_rot, ':', 'b')
                plot_line((lev_arm_vec_r + Poi_center_rot), Poi_center_rot, ':', 'r')
                plot_line((lev_arm_vec_f + Poi_center_rot), Poi_center_rot, ':', col_green)
            end

            % plot gravity force
            plot_force_vec(Pat_CoM_lean, F_grav_vec, '-', 'c', lbl_F_g, 'left', FORCE_SCALING/7, 0)
        end


        % net of forces acting on harness (total, parasitic, movement generating)
        % plot_force_vec(har_c_lean, fnet_harn, '-', "k", 'F_{har net}', 'left', FORCE_SCALING, 0)
        % plot_force_vec(har_c_lean, fnet_harn_para, '-', [0.6350 0.0780 0.1840], 'F_{har para}', 'left', FORCE_SCALING, 0)
        % plot_force_vec(har_c_lean, fnet_harn_move, '-', [0.4660 0.6740 0.1880], 'F_{har move}', 'left', FORCE_SCALING, 0)

        if flag_SHOW.torque_gen_forces
            % change name based on if the magniutde should be included or not
            if flag_SHOW.vector_magnitude
                lbl_F_ltgen = strcat('F_{l-tgen}=', num2str(norm(fvec_comp_l_move)));
                lbl_F_rtgen = strcat('F_{r-tgen}=', num2str(norm(fvec_comp_r_move)));
                lbl_F_ftgen = strcat('F_{f-tgen}=', num2str(norm(fvec_comp_f_move)));
                lbl_F_gtgen = strcat('F_{g-tgen}=', num2str(norm(fvec_comp_g_move)))
```

```
                ;
        else
            lbl_F_ltgen = 'F_{l-tgen}';
            lbl_F_rtgen = 'F_{r-tgen}';
            lbl_F_ftgen = 'F_{f-tgen}';
            lbl_F_gtgen = 'F_{g-tgen}';
        end

        % plot torque generating forces
        plot_force_vec(har_l_lean, fvec_comp_l_move, '--', 'b', lbl_F_ltgen, 'left', FORCE_SCALING, 0)
        plot_force_vec(har_r_lean, fvec_comp_r_move, '--', 'r', lbl_F_rtgen, 'left', FORCE_SCALING, 0)
        plot_force_vec(har_f_lean, fvec_comp_f_move, '--', col_green, lbl_F_ftgen, 'left', FORCE_SCALING, 0)
        plot_force_vec(Pat_CoM_lean, fvec_comp_g_move, '--', 'c', lbl_F_gtgen, 'left', FORCE_SCALING, 0)
    end

    if flag_SHOW.parasitic_forces
        % change name based on if the magniutde should be included or not
        if flag_SHOW.vector_magnitude
            lbl_F_lpara = strcat('F_{l-para}=', num2str(norm(fvec_comp_l_move)));
            lbl_F_rpara = strcat('F_{r-para}=', num2str(norm(fvec_comp_r_move)));
            lbl_F_fpara = strcat('F_{f-para}=', num2str(norm(fvec_comp_f_move)));
        else
            lbl_F_lpara = 'F_{l-para}';
            lbl_F_rpara = 'F_{r-para}';
            lbl_F_fpara = 'F_{f-para}';
        end

        % plot parasitic forces
        plot_force_vec(har_l_lean, fvec_comp_l_para, '-.', 'b', lbl_F_lpara, 'right', FORCE_SCALING, 0)
        plot_force_vec(har_r_lean, fvec_comp_r_para, '-.', 'r', lbl_F_rpara, 'right', FORCE_SCALING, 0)
        plot_force_vec(har_f_lean, fvec_comp_f_para, '-.', col_green, lbl_F_fpara, 'right', FORCE_SCALING, 0)
    end



    if flag_SHOW.moments
        % change name based on if the magniutde should be included or not
        if flag_SHOW.vector_magnitude
            lbl_F_lmom = strcat('F_{l-mom}=', num2str(norm(fvec_comp_l_move)));
            lbl_F_rmom = strcat('F_{r-mom}=', num2str(norm(fvec_comp_r_move)));
            lbl_F_fmom = strcat('F_{f-mom}=', num2str(norm(fvec_comp_f_move)));
        else
            lbl_F_lmom = 'F_{l-mom}';
            lbl_F_rmom = 'F_{r-mom}';
            lbl_F_fmom = 'F_{f-mom}';
        end
        % plot moments acting on child
        plot_force_vec(har_l_lean, fvec_comp_l_moment, ':', 'b', lbl_F_lmom, 'center', FORCE_SCALING, 0)
```

```
            plot_force_vec(har_r_lean, fvec_comp_r_moment, ':', 'r', lbl_F_rmom, 'center', FORCE_SCALING, 0)
            plot_force_vec(har_f_lean, fvec_comp_f_moment, ':', col_green, lbl_F_fmom, 'center', FORCE_SCALING, 0)
        end


        if flag_SHOW.show_field
            twist_ang = Vars.twist;
            plotForceFieldHemisphere(0, 0, 9, 11, twist_ang, Vars, Child, R_INTERCON, flag_SHOW, Boundary, "", false, true);
        end

    end




    %% output data
    % resulting torque related information
    output_data.t_net_vec = tnet_vec;                     % (Nm) net torque vector
    output_data.t_net_mag = tnet_mag;                     % (Nm) magnitude of net torque (no sign)
    %output_data.t_net_direc = direction_torque;          % (-1 to 1) difference between net torque and ideal torque direction based on projection
    output_data.t_net_direc = ang_tdirec_dif;             % (deg) difference between net torque and ideal torque direction based on angle between two vectors
    %output_data.t_ang_dif_from_ideal = ang_tdirec_dif;  % (deg) difference between net torque and ideal torque direction based on angle between two vectors
    output_data.har_c = har_c_lean;                       % (m) 3D point vector of center of harness when leaning
    output_data.center_xy = Har_C_0(1:2);                 % (m) 2D point vector of central position of device


    % parasitic force information
    output_data.f_mag_para_max = f_mag_para_max;         % (N) largest parasitic force magnitude
    output_data.f_mag_para_l = mag_Fl_comp_para;         % (N) parasitic force on left harness point
    output_data.f_mag_para_r = mag_Fr_comp_para;         % (N) parasitic force on right harness point
    output_data.f_mag_para_f = mag_Ff_comp_para;         % (N) parasitic force on front harness point

    % return tension force in each rope
    output_data.f_mag_rope_max = fmag_rope_max_end;      % (N) maximum tension experienced by ropes
    output_data.f_mag_rope_lr = fmag_LR_end;             % (N) left(-right) rope tension
    output_data.f_mag_rope_rf = fmag_RF_end;             % (N) right(-front) rope tension
    output_data.f_mag_rope_fl = fmag_FL_end;             % (N) front(-left) rope tension

    % these track if there is slack in the rope
    output_data.flg_no_slk_lr = Flag_no_slack_lr;
    output_data.flg_no_slk_rf = Flag_no_slack_rf;
    output_data.flg_no_slk_fl = Flag_no_slack_fl;
```

```

    % other checks  --- If "true" then good, if "false" then bad ---
    %output_data.flg_arm_safe = Flag_arm_safe;
    output_data.flg_trunk_len = Flag_trunk_len_good;

    % these track if rope stretching is good  --- If "true" then good, if "false"
        then bad ---
    output_data.unstretched_rope_length = Len_Cab_LR_Unstr;      % defines how long
        of a rope should be purchaced
    output_data.flg_r_unstrchlen = Flag_rope_unstretch_good;     % bad if unstrethced
         rope length <= 0
    output_data.flg_r_strchprop_0 = Flag_rope_stretch_0_good;
    output_data.flg_r_strchprop_lr_end = Flag_rope_stretch_lr_1_good;
    output_data.flg_r_strchprop_rf_end = Flag_rope_stretch_rf_1_good;
    output_data.flg_r_strchprop_fl_end = Flag_rope_stretch_fl_1_good;
    output_data.r_strch_perc_0 = r_stretch_perc_0;
    output_data.r_strch_perc_max = r_stretch_perc_end_max;
end
```

F.3 Optimization Code

```
%% OPTIMIZATION FUNCTION
function [output_data, flags_Optim_return] = optim_genetic_with_cost_func(Vars,
    Child, R_INTERCON, ...
    flag_SHOW, Boundary, positions_of_interest, size_of_pop, num_top_sol,
        num_fail_gen, size_mutation)
% Cost function aims to be as LOW as possible (minimizing). Uses genetic
    optimization to do this
% positions_of_interest =

    % tracking variables
    same_best = 0;            % number of generations a solution has lasted for
    gen_num = 1;              % track number of generations examined
    max_num_gen = 600;        % sets number of allowed generations (preventing
        infinite loops)
    num_no_sol = 0;           % used to track number of times no good solutions were
        found
    num_no_sol_count = 0;     % tracks the number of times a no solution condition is
        triggered

    num_of_inputs = 3;  % this will need to change if more variables to tweak are
        added


    % will save previous solution (start at infinity to guarantee success)
    % [gen      individual      cost]
    prev_sol = [0 0 Inf];


    % create array that will store each generation created and creat the first
        generation
    gen = NaN(num_fail_gen, size_of_pop, num_of_inputs);
    %   gen(generation number, individual solution, value being changed)
    %                               [K_rope      Pre_tension      Upper_pully_offset]
    for indiv=1:size_of_pop
        gen(gen_num, indiv, :) = [make_rand_num(Boundary.r_k_min, Boundary.r_k_max,
            'd') make_rand_num(Boundary.r_preten_min, Boundary.r_preten_max, 'd')
            make_rand_num(Boundary.p_offset_min, Boundary.p_offset_max, 'd')];
    end

```

```

    % create some empty arrays of the minimum size to help with performance (will likely still change in size)
    gen_sim_tmags = NaN(num_fail_gen, size_of_pop);
    gen_sim_tdirecs = NaN(num_fail_gen, size_of_pop);
    gen_sim_fparas = NaN(num_fail_gen, size_of_pop);
    V_cost = NaN(num_fail_gen, size_of_pop);
    gen_weight = NaN(num_fail_gen, size_of_pop);

    % Loop until a solution has lasted for given number of generations (with generation limit and no good solutions limit)
    while (same_best < num_fail_gen  || prev_sol(1) == 0) && gen_num < max_num_gen && num_no_sol < 3

        if mod(gen_num, 20) == 0
            % just prints information about time elapsed
            track_time_elapsed('      Genetic optimization generation', gen_num, 0)
        end


        % find solutions for each generation's individuals at the given points
        for indiv=1:size_of_pop
            Vars.r_k = gen(gen_num, indiv, 1);
            Vars.r_preten = gen(gen_num, indiv, 2);
            Vars.p_offset = gen(gen_num, indiv, 3);

            for position_check = 1:size(positions_of_interest, 1)
                Vars.l_medlat = positions_of_interest(position_check, 1);
                Vars.l_antpos = positions_of_interest(position_check, 2);
                Vars.twist = positions_of_interest(position_check, 3);

                gen_sim_out = sim_device(Vars, Child, R_INTERCON, flag_SHOW, Boundary);

                % save key outputs of simulation
                gen_sim_tmags(gen_num, indiv, position_check) = gen_sim_out.t_net_mag;
                gen_sim_tdirecs(gen_num, indiv, position_check) = gen_sim_out.t_net_direc;
                gen_sim_fparas(gen_num, indiv, position_check) = gen_sim_out.f_mag_para_max;

                % save the checks on rope stretching
                gen_sim_flags(gen_num, indiv, position_check).flg_r_unstrchlen = gen_sim_out.flg_r_unstrchlen;
                gen_sim_flags(gen_num, indiv, position_check).flg_r_strchprop_0 = gen_sim_out.flg_r_strchprop_0;
                gen_sim_flags(gen_num, indiv, position_check).flg_r_strchprop_lr_end = gen_sim_out.flg_r_strchprop_lr_end;
                gen_sim_flags(gen_num, indiv, position_check).flg_r_strchprop_rf_end = gen_sim_out.flg_r_strchprop_rf_end;
                gen_sim_flags(gen_num, indiv, position_check).flg_r_strchprop_fl_end = gen_sim_out.flg_r_strchprop_fl_end;

            end
        end

        % give cost function:
        % for all individuals, results of simulations at each position
```

```
        % the given positions of interest
        % flags of anomalies from the code
        % size of the population (so it doesn't need to be calculated again)
        gen_manip_values = squeeze(gen(gen_num, :, :));     % information for each individual
        tmags = squeeze(gen_sim_tmags(gen_num, :, :));      % torque magnitudes per individual
        tdirecs = squeeze(gen_sim_tdirecs(gen_num, :, :));  % torque direction per individual
        fparas = squeeze(gen_sim_fparas(gen_num, :, :));    % largest parasitic force per individual
        flags = squeeze(gen_sim_flags(gen_num, :, :));      % flags of simulation

        % gets array of the cost function value for each individual
        [V_cost(gen_num, :), flags_Optim(gen_num, :)] = cost_function(gen_manip_values, Child.t_opp_str, tmags, tdirecs, fparas, flags, size_of_pop);



        % find the best n solutions and their respective individual (lower is better)
        [best_n_sol{gen_num}, best_n_indiv{gen_num}] = mink(V_cost(gen_num, :), num_top_sol);


        % check if the new solution is better (lower) than the old
        if best_n_sol{gen_num}(1) < prev_sol(3)       % if new best solution better than old

            % save info on this new best solution
            prev_sol = [gen_num best_n_indiv{gen_num}(1) best_n_sol{gen_num}(1)];

            same_best = 0;   % reset as new better solution is found

        else             % if new best solution is not better than old
            same_best = same_best+1;
        end


        % check if more individuals need to be created (bit of a bandaid solution for the ordering issue)
        if same_best < num_fail_gen || prev_sol(1) == 0

            % === start creating new set of individuals ===
            % top "num_top_sol" individuals are kept the same (Elitist)
            for indiv=1:num_top_sol
                gen(gen_num+1, indiv, :) = gen(gen_num, best_n_indiv{gen_num}(indiv), :);
            end


            % --- create children with parents (Crossover section)

            % check to see if all values are Inf
            if all(abs(V_cost(gen_num, :)) == Inf)
                % means no individual has a good solution
                num_no_sol = num_no_sol+1;
                num_no_sol_count = num_no_sol_count+1;
```

```
            gen_num = gen_num-1;        % retry the generation to hopefully get better values
            fprintf(2, "No solution for generation %d could not be found. Retrying", gen_num)

        else    % else the code should work fine
            % how this section works could be sped up a lot

            num_no_sol = 0;       % reset this

            % tracks number of individuals that are not Inf
            % necessary in the case that only 1 exists
            num_of_valid_sols = 0;

            % find maximum value that is not -Inf. Necessary as Inf values mess things up
            max_V_cost = -Inf;
            for g=1:size_of_pop
                if V_cost(gen_num, g) > max_V_cost && V_cost(gen_num, g) ~= Inf
                    max_V_cost = V_cost(gen_num, g);    % largest cost that isn't Inf
                end

                % track number of valid solutions that are not Inf
                if V_cost(gen_num, g) ~= Inf
                    num_of_valid_sols = num_of_valid_sols+1;
                end
            end

            % set weighting (0-1) each individual gets based on their score compared to largest.
            % Individuals of lower scores are more likely to be selected
            for g=1:size_of_pop
                if abs(V_cost(gen_num, g)) == Inf   % incase any negatives appear somehow
                    % these values are known bad and should not be included in weighting (= 0)
                    gen_weight(gen_num, g) = 0;

                else
                    if num_of_valid_sols > 1
                        % these values are good and should be compared to max. Lower the V_cost
                        % the higher the weighting (more likely to be picked)
                        gen_weight(gen_num, g) = 1 - V_cost(gen_num, g)/max_V_cost;

                    else    % in this case the only valid option has been found
                        % set it as the most likely option
                        gen_weight(gen_num, g) = 1;

                    end
                end
            end
        end


        % create parents based on weighting
        parent_1 = randsample(1:size_of_pop, size_of_pop-num_top_sol, true,
```

```
            gen_weight(gen_num, :));
        parent_2 = randsample(1:size_of_pop, size_of_pop-num_top_sol, true, gen_weight(gen_num, :));


        % Crossover (parents' genetics get mixed)
        for couple=1:size_of_pop-num_top_sol
            % determines where to split genes between parents
            where_to_split = make_rand_num(1, num_of_inputs-1, 'i');

            % create child by combining sections of parents' genes
            gen(gen_num+1, couple+num_top_sol, :) = [squeeze(gen(gen_num, parent_1(couple), 1:where_to_split))' squeeze(gen(gen_num, parent_2(couple), where_to_split+1:num_of_inputs))'];
        end


        % mutate each individual slightly
        for indiv=1:size_of_pop
            which_to_mutate = make_rand_num(1, num_of_inputs, 'i');


            % based on which value to mutate, save appropreate boundary values
            if which_to_mutate == 1
                new_val_max = Boundary.r_k_max;
                new_val_min = Boundary.r_k_min;

            elseif which_to_mutate == 2
                new_val_max = Boundary.r_preten_max;
                new_val_min = Boundary.r_preten_min;

            elseif which_to_mutate == 3
                new_val_max = Boundary.p_offset_max;
                new_val_min = Boundary.p_offset_min;

            else    % some kind of big error
                return

            end


            % Calculate the maximum change in value
            maxChange = (new_val_max-new_val_min)*size_mutation/100;

            % Generate a random number within the range of (startValue +- maxChange)
            new_val = gen(gen_num+1, indiv, which_to_mutate) + maxChange*(2*rand() - 1);

            % Ensure the generated number is within the given boundaries
            new_val = max(min(new_val, new_val_max), new_val_min);


            % old way of mutating a number, may not need if upper one is working fine
            % % set the starting new_val value to something outside the bounds. Allows loop to begin
            % new_val = new_val_max * 2;
            %
```

```
                %
                % % loop until the new value fits within the bounds
                % while new_val<=new_val_min || new_val>=new_val_max
                %
                %     % create mutation percentage
                %     mutation = make_rand_num(0, size_mutation/100, 'd');
                %
                %     % get size of that mutation percent based on range of that value
                %     per_of_val = (abs(new_val_min) + abs(new_val_max))*mutation;
                %
                %     % randomly pick if the change should increase or decrease original value
                %     if make_rand_num(0, 1, 'i') == 1
                %         new_val = gen(gen_num+1, indiv, which_to_mutate) + per_of_val;
                %     else
                %         new_val = gen(gen_num+1, indiv, which_to_mutate) - per_of_val;
                %     end
                %
                % end

                % set gene component to new value
                gen(gen_num+1, indiv, which_to_mutate) = new_val;
            end

            gen_num = gen_num+1;      % increment generation by 1
        end
    end


    % save global info
    output_data.num_no_sol_count = num_no_sol_count;

    if num_no_sol < 3
        % set up information that will be returned
        output_data.num_gen_best = prev_sol(1);      % generation in which maximum was found
        output_data.v_cost_val = prev_sol(3);        % cost function result of the best generation

        % setup information of best solution
        output_data.r_k = gen(prev_sol(1), prev_sol(2), 1);
        output_data.r_preten = gen(prev_sol(1), prev_sol(2), 2);
        output_data.p_offset = gen(prev_sol(1), prev_sol(2), 3);
        flags_Optim_return = flags_Optim(prev_sol(1), prev_sol(2));

    else
        disp("No solutions could be found after 3 attempts. Rethink cost function")
        output_data.v_cost_error = V_cost;
        return
    end

    % if the generation limit was not hit and solutions were found then the optimization did
    % converge to something
    if gen_num < max_num_gen && num_no_sol < 3
        flags_Optim_return.converged = true;
```

```
    else
        flags_Optim_return.converged = false;   % flag convergence as unsuccessful
    end
end
```

F.4 Cost Function

```
function [V_cost, flags_Optim] = cost_function(gen_manip_values, child_oppos, tmags, tdirecs, fparas, flags, size_of_pop)
% given the inputs to the cost funciton, the resulting value is returned. ONLY looks at 1 generation
% Cost function aims to be as LOW as possible
    child_oppos_min = child_oppos*1.1;
    child_oppos_max = child_oppos*1.2;

    % create array so that it doesn't need to keep changing size
    V_cost = squeeze(nan(1, size_of_pop));

    % find cost function for each individual
    for indiv=1:size_of_pop
        % used to influence the whole cost function based on results of flags
        global_multiplier = 1;          % 1 means function does not need to be influenced


        % if the rope stretching is outside given rope spec, then this is a problem
        if all([flags(indiv, :).flg_r_unstrchlen flags(indiv, :).flg_r_strchprop_0 ...
                flags(indiv, :).flg_r_strchprop_lr_end flags(indiv, :).flg_r_strchprop_rf_end ...
                flags(indiv, :).flg_r_strchprop_fl_end] == 1)

            flags_Optim(indiv).flg_check = -1;     % flags are fine

        else
            % something is not right with one or more of the ropes, penalise heavily and still allow
            global_multiplier = Inf;
            flags_Optim(indiv).flg_check = 1;      % flags are NOT fine
        end



        % logically check magnitude of first arrow boundary
        tmag_min = min(tmags(indiv, :));
        if tmag_min < child_oppos_min          % child support too weak, not allowed
            global_multiplier = Inf;
            flags_Optim(indiv).tmag = 1;  % saves info about logic outcomes of individual (bad)

        elseif tmag_min > child_oppos_max   % too strong for child
            global_multiplier = global_multiplier*100;
            flags_Optim(indiv).tmag = 0;      % saves info about logic outcomes of individual (not ideal)

        else
            % give incentive for being in ideal range
            global_multiplier = global_multiplier/10;
            flags_Optim(indiv).tmag = -1;   % saves info about logic outcomes of individual (very good)
```

```
        end



        % logically check direction of worst arrow
        max_tdirec = max(tdirecs(indiv, :));
        if max_tdirec < 5
            % arrows are aimed super well towards center. give incentive
            global_multiplier = global_multiplier/10;
            flags_Optim(indiv).tdirec = -1;      % saves info about logic outcomes of individual (very good)

        elseif max_tdirec > 20
            % arrows don't point towards the center as intended. Not allowed (this would also include no support)
            global_multiplier = Inf;
            flags_Optim(indiv).tdirec = 1;      % saves info about logic outcomes of individual (bad)

        else
            % direction is not great
            global_multiplier = global_multiplier*100;

            flags_Optim(indiv).tdirec = 0;      % saves info about logic outcomes of individual (fine)

        end



        max_fpara = max(abs(fparas(indiv, :)));  % get maximum parasitic force magnitude to compare

        % final cost funciton. lower is better
        V_cost(indiv) = global_multiplier*(gen_manip_values(indiv, 1) + gen_manip_values(indiv, 2) + max_fpara^2);
    end
end
```

F.5 Patient Class

```
% Class used to stores unchanging details about the patient
classdef Patient
    properties
        age {mustBeNumeric}         % (years)
        hei {mustBeNumeric}         % (m) total height of child
        wei {mustBeNumeric}         % (kg) total weight of child
        w_cir {mustBeNumeric}       % (m) waist circumference (equivalent to harness length)
        t_rad {mustBeNumeric}       % (m) trunk radius (&&& assuming trunk is cylinder)

        tip_head {mustBeNumeric}    % (m) length from buttox to tip of head when seated upright
        arm_len {mustBeNumeric}     % (m) length of arms fully extended
        t_len {mustBeNumeric}       % (m) length from buttox to tip of shoulders when seated upright
```

```
        CoM {mustBeNumeric}          % (m) location of center of mass, along torso, of child when seated
        chair {mustBeNumeric}        % (m) height of chair from ground to top of seat
        t_wei_prop {mustBeNumeric}   % (Unit) proportion of body weight that is part of torso

        t_opp_str {mustBeNumeric}  % (Nm) how much torque the child is able to provide
    end

    methods
        % constructor
        function obj = Patient(age, height, weight, waist_circum, opposition)
            obj.age = age;
            obj.hei = height;
            obj.wei = weight;

            obj.w_cir = waist_circum;
            obj.t_rad = waist_circum/2/pi();

            % https://www.sciencedirect.com/science/article/pii/0021929086900126
            obj.t_wei_prop = 0.65;   % &&& guess of how much weight is in the torso (~65%)


            % ----- patient dimention setting -----
            % based on taking the 95th or 5th percentile of the data and generating percentages based on height
            % https://dined.io.tudelft.nl/en/database/tool

            % data for 8 year old males in 95th percentile
            hei_stand_8m = 1.422;                % (m) standing height
            hei_sit_head_8m = 0.753;             % (m) buttox to top of head
            hei_sit_should_8m = 0.468;           % (m) database does not have buttox to armpit
            hei_seat_8m = 0.401;                 % (m) bottom of feet to buttox level (90 deg knees)
            arm_len_8m = 1.039;                  % (m) length of arms [labled reach depth on dined]


            % data for 3 year old females in 5th percentile
            hei_stand_3f = 0.93;                 % (m) standing height
            hei_sit_head_3f = 0.528;             % (m) buttox to top of head
            hei_sit_should_3f = 0.315;           % (m) database does not have buttox to armpit
            hei_seat_3f = 0.219;                 % (m) bottom of feet to buttox level (90 deg knees)
            arm_len_3f = 0.66;                   % (m) length of arms [labled reach depth on dined]


            % linear extrapolation for other measures
            obj.arm_len = linearExtrap(3, 8, arm_len_3f, arm_len_8m, age);

            % min an maximum age values
            % base proportions on the age given
            prop_hei_stand = linearExtrap(3, 8, hei_stand_3f, hei_stand_8m, age);                     % (m) standing height
```

```
            prop_hei_sit_head = linearExtrap(3, 8, hei_sit_head_3f, hei_sit_head_8m,
                 age);       % (m) buttox to top of head
            prop_hei_sit_should = linearExtrap(3, 8, hei_sit_should_3f,
                hei_sit_should_8m, age);% (m) database does not have buttox to
                armpit
            prop_hei_seat = linearExtrap(3, 8, hei_seat_3f, hei_seat_8m, age);
                                    % (m) bottom of feet to buttox level (90 deg knees)

            % get percentage of overall height each measure takes up
            perc_head = prop_hei_sit_head/prop_hei_stand;
            perc_torso = prop_hei_sit_should/prop_hei_stand;
            perc_chair = prop_hei_seat/prop_hei_stand;

            % get length compared to height of child
            obj.tip_head = height * perc_head;
            obj.t_len = height * perc_torso;
            obj.chair = height * perc_chair;


            % location of COM of -ADULTS- (couldn't find children) with respect to
                trunk length from bottom
            % https://jestec.taylors.edu.my/Vol%2011%20issue%202%20February%202016/
                Volume%20(11)%20Issue%20(2)%20166-%20176.pdf
            % source has between 49.5-56.2 (pg. 4) = 0.5285
            % from the course = 0.626
            obj.CoM = obj.t_len * 0.626;

            % using the maximum for males as the base so we can see maximum forces
            % values of trunk force found in:
            % https://www.sciencedirect.com/science/article/pii/S002561961163276X#
                fig2
            opp_force_m6_max = 220.5;        % (N)
            opp_force_m8_max = 273.9;

            % Relationship between trunk muscle strength, reaching ability and
                balance in children with Down syndrome - A cross-sectional study
            opp_force_max_107_ds = 4.21*9.81;      % (N after calculation)
            opp_force_max_107_td = 5.93*9.81;


            if opposition < 0
                opp_force = linearExtrap(6, 8, opp_force_m6_max, opp_force_m8_max,
                    age);
                obj.t_opp_str = opp_force*obj.t_len;

                % https://www.tandfonline.com/doi/pdf/10.1080/13638490310001654754
                t_opp_str_m_max_12_nbp = 148.5;         % (Nm)
                t_opp_str_m_max_12_bp = 156.9;

                % https://www.sciencedirect.com/science/article/pii/
                    S0161475415000755
                opp_force_10_np = 112.8;         % (Nm)
                opp_force_11_np = 147.7;
                opp_force_10_ip = 106.3;
                opp_force_11_ip = 133.4;


            elseif opposition == 0
                % estimate strength based on weight  &&& will need to change 40 if
```

```
                    boundaries change
                obj.t_opp_str = estOppositionStrength(weight, obj.t_wei_prop, obj.CoM, 40);

            else
                obj.t_opp_str = opposition;
            end
        end
    end
end
```

F.6 Mechanical Requirements

```
% use this code to find some mechanical loads the device would need to be able to withstand.
% ===== assumes an output_data.mat file has been added or the required data is alreaedy in memory
clc
format longg
setBoundary

%% calculate everything
% create variables
max_lr_rope_ten = zeros(2, 2);
max_rf_rope_ten = zeros(2, 2);
max_fl_rope_ten = zeros(2, 2);
max_rope_ten = zeros(2, 2);
max_pulley_load = zeros(2, 2);
rope_k = zeros(2, 2);
v = zeros(2, 2);

% create extreme ends for children
Child_max = Patient(Boundary.pat_age_max, Boundary.pat_hei_max, Boundary.pat_wei_max, Boundary.pat_waist_max, -1);
Child_min = Patient(Boundary.pat_age_min, Boundary.pat_hei_min, Boundary.pat_wei_min, Boundary.pat_waist_min, -1);

% maximum height the column needs to be for 8 year old + a safety factor of 10cm
col_l_max = Child_max.chair + Child_max.t_len*Boundary.harn_h_max/100 + Boundary.p_offset_max;
col_l_max_safe = col_l_max + 0.1;

[num_rope_configs, num_children] = size(output_data);   % get dimentions of the array

% structural information about the columns
% https://uk.rs-online.com/web/p/tubing-and-profile-struts/4667219
I = 0.7/10^4;                   % (m^4) moment of intertia
E_alum = 68.9*10^9;             % (Pa) https://www.engineeringtoolbox.com/properties-aluminum-pipe-d_1340.html
safety_factor = 2;


% loop though each combination to find important values
for rope_config = 1:num_rope_configs        % 1=interconnected, 2=separate
    for child_test = 1:num_children          % 1=3YO, 2=8YO
        % Find forces on pulleys
        max_lr_rope_ten(rope_config, child_test) = max([output_data(rope_config, child_test).sim_results.f_mag_rope_lr]);
```

```
        max_rf_rope_ten(rope_config, child_test) = max([output_data(rope_config, child_test).sim_results.f_mag_rope_rf]);
        max_fl_rope_ten(rope_config, child_test) = max([output_data(rope_config, child_test).sim_results.f_mag_rope_fl]);

        max_rope_ten(rope_config, child_test) = max([max_lr_rope_ten(rope_config, child_test) max_rf_rope_ten(rope_config, child_test) max_fl_rope_ten(rope_config, child_test)]);

        if rope_config == 1
            max_pulley_load(rope_config, child_test) = max_rope_ten(rope_config, child_test)*2;

        else
            max_pulley_load(rope_config, child_test) = max_rope_ten(rope_config, child_test);

        end


        % Find minimum stiffness of rope needed
        rope_k(rope_config, child_test) = output_data(rope_config, child_test).gen_optim(top_n_indiv(rope_config, child_test)).r_k;


        % find diflection of column (just considering top pulley as bottom won't affect much)
        % https://www.structuralbasics.com/beam-deflection-formulas/#cantilever-beam-intermediate-point-load-
        % https://mechanicalc.com/reference/beam-deflection-tables
        if output_data(rope_config, child_test).r_config    % config with interconnected ropes
            % have to *2 since there could be 2 ropes acting at the top of the column. Assuming
            % worst case that they are both max (though this is unrealistic)

            v(rope_config, child_test) = safety_factor*((max_rope_ten(rope_config, child_test)*2*col_l_max^3)/(6*E_alum*I))*(3*col_l_max_safe - col_l_max)*1000;

        else        % config with separate ropes
            v(rope_config, child_test) = safety_factor*((max_rope_ten(rope_config, child_test)*col_l_max^3)/(6*E_alum*I))*(3*col_l_max_safe - col_l_max)*1000;

        end
    end
end


%% print everything

% minimum column length
fprintf("Minimum column length = \t\t\t\t\t%g m\n\n", col_l_max_safe);

% pulley loading
max_pulley_load_inter = max(max_pulley_load(1, :), [], "all");
max_pulley_load_sep = max(max_pulley_load(2, :), [], "all");
fprintf("Maximum pulley load (interconnected) = \t\t%g N\n", max_pulley_load_inter);
```

```
fprintf("Maximum pulley load (separate) = \t\t\t%g N\n\n", max_pulley_load_sep);

% rope information
max_rope_ten_inter = max(max_rope_ten(1, :), [], "all");
min_rope_k_inter = min(rope_k(1, :), [], "all");
max_rope_k_inter = max(rope_k(1, :), [], "all");
max_rope_ten_sep = max(max_rope_ten(2, :), [], "all");
min_rope_k_sep = min(rope_k(2, :), [], "all");
max_rope_k_sep = max(rope_k(2, :), [], "all");

fprintf("Maximum rope tension (interconnected) = \t%g N\n", max_rope_ten_inter);
fprintf("Minimum rope stiffness (interconnected) = \t%g N/m\n", min_rope_k_inter);
fprintf("Maximum rope stiffness (interconnected) = \t%g N/m\n", max_rope_k_inter);
fprintf("Maximum rope tension (separate) = \t\t\t%g N\n", max_rope_ten_sep);
fprintf("Minimum rope stiffness (separate) = \t\t%g N/m\n", min_rope_k_sep);
fprintf("Maximum rope stiffness (separate) = \t\t%g N/m\n\n", max_rope_k_sep);

disp("Interconnected ropes")    % rope_config = 1 = true
fprintf("Unstretched rope length for smallest 3 year old child = %g m\n", ...
    output_data(1, 1).sim_results(1, 1).unstretched_rope_length);
fprintf("Max starting rope stretch percentage for smallest 3 year old child = %g %% \n", ...
    max([output_data(1, 1).sim_results.r_strch_perc_0]));
fprintf("Max overall rope stretch percentage for smallest 3 year old child = %g %% \n", ...
    max([output_data(1, 1).sim_results.r_strch_perc_max]));
fprintf("Unstretched rope length for largest 8 year old child = \t%g m\n", ...
    output_data(1, 2).sim_results(1, 1).unstretched_rope_length);
fprintf("Max starting rope stretch percentage for smallest 8 year old child = %g %% \n", ...
    max([output_data(1, 2).sim_results.r_strch_perc_0]));
fprintf("Max overall rope stretch percentage for smallest 8 year old child = %g %% \n\n", ...
    max([output_data(1, 2).sim_results.r_strch_perc_max]));

disp("Separate ropes")          % rope_config = 2 = false
fprintf("Unstretched rope length for smallest 3 year old child = %g m\n", ...
    output_data(2, 1).sim_results(1, 1).unstretched_rope_length);
fprintf("Starting rope stretch percentage for smallest 3 year old child = %g %% \n", ...
    max([output_data(2, 1).sim_results.r_strch_perc_0]));
fprintf("Max overall rope stretch percentage for smallest 3 year old child = %g %% \n", ...
    max([output_data(2, 1).sim_results.r_strch_perc_max]));
fprintf("Unstretched rope length for largest 8 year old child = \t%g m\n", ...
    output_data(2, 2).sim_results(1, 1).unstretched_rope_length);
fprintf("Max starting rope stretch percentage for smallest 8 year old child = %g %% \n", ...
    max([output_data(2, 2).sim_results.r_strch_perc_0]));
fprintf("Max overall rope stretch percentage for smallest 8 year old child = %g %% \n\n", ...
    max([output_data(2, 2).sim_results.r_strch_perc_max]));

% max column displacement (mm)
v_max_inter = max(v(1, :), [], "all");
v_max_sep = max(v(2, :), [], "all");
fprintf("Maximum column displacement (interconnected) = \t%g mm\n", v_max_inter);
fprintf("Maximum column displacement (separate) = \t\t%g mm\n", v_max_sep);
```