MechaHorse 2.0

Final Report

Team Members

Yap Chun Lin

Jill Ueng

Yann Zhong

Chan KwanKiu

Luka Lagator

Timothee Gathmann

Piramol Krishnan

Emma Duncanova

Magdalene Ho

Supervisor: Dr. Warren MacDonald

Revision	Date	Author	Description
A	08/06/2017	Various	First release

8 June 2017

Word Count: 5164

Abstract

The primary objective of this project was to improve on the anterior design of the MechaHorse, an equine riding simulator designed to provide a user friendly entry point into horse-riding for mentally disabled users. The MechaHorse was designed to be a low cost therapy tool that simulated the motion of an actual horse, thereby gauging a rider's competency to progress safely to actual horse riding.

The MechaHorse aims to simulate a steady circular "walk" motion and a faster elliptical "trot" motion. The motion is generated by a crankshaft, powered by a motor and controlled through a user friendly app that operates on any smartphone. This design improves upon various points as of its predecessor: the main one being the addition of an additional gait and an easily operable user interface. Attempts at motorization have also been brought a long way, only hindered by a lack of torque in the provided motor. The additional purchase of a spring, that offsets load upon the motor, allows for operation for a wider range of weight of riders. Verification of the efficiency and accuracy of the MechaHorse is tested through the use of a pixel tracking program, applied to various test subjects. Potential improvements are also discussed in detail in the body of the report. The appendix feature schematics, calculations, codes and additional information for a more fundamental understanding of the MechaHorse.

Contents

1. Introduction 1

1.1. Aims 1

1.2. Objectives 1

1.3. Background Research 1

1.4. User Requirements 2

1.5. Technical Requirements 2

1.6. Cost Requirements 2

2. Final Design 3

2.1. Overview 3

2.2. Description of the components 4

2.2.1. Actuator Housing Unit and Actuators 4

2.2.2. Saddle Support Unit 5

2.2.3. Crankshaft 5

2.2.4. Motor 6

2.2.5. Drive unit 6

2.2.6. Spring assembly 6

2.2.7. Additional safety mechanisms 7

2.2.8. Electrical components box 8

2.2.9. Circuit 8

2.2.10. Software 9

2.3. Prototype 10

3. Testing and Evaluation 11

3.1. Physical testing 11

3.2. Computational Analysis 13

4. Discussion 15

5. **Conclusion** 18

6. References 19

Appendix A – Project Management 20

Appendix B – Risk Management 21

Appendix C – Manufacturing 24

Appendix D – Bill of Materials 25

Appendix E – Initial Ideas and Design 27

Appendix F – User Manual 31

Appendix G – Arduino Code 32

Appendix H – Prototypes and Rough Specifications of Components 35

Appendix I – Stress Analysis 38

Appendix J – Ethics 40

Appendix K – Acknowledgements 41

1. Introduction

Equine therapy has been popular for its therapeutical effects such as stress relief, confidence building and body coordination development[1]. However, riding for the disabled (RDA) schools have difficulty affording such facilities and benefits as their shared budget of £76,000 across 470 institutions is minute compared to the minimum cost of £19,680 for current on-market horse. The MechaHorse aims to provide a means to develop the user's confidence and core muscles prior to riding a horse to ensure maximum safety. The design of the MechaHorse is focused on the mentally disabled as there are too many considerations that have to be taken into account for the physically disabled, with very diverse and specific needs, that cannot be achieved within the timeframe of our project.

The primary framework of the MechaHorse was inherited from the previous group. It was hand cranked and had improvements to be made to the simulated motion of the horse walk motion. Hence, development to the MechaHorse have since been made, which includes the correction of the walking gait motion and addition of another gait motion, trotting. Motorization of the MechaHorse was also explored, with further details in the Final Design and Discussion.

1.1. Aims

The aim of this project is to design and produce a safe and manufacturable motorized horse simulator which provides two motions of a horse: walking and trotting through a user-friendly app control interface. The MechaHorse's primary purpose is to act as a test for disabled users prior to getting on a real horse. Only when the user is comfortable with the motions can he/she proceed to engaging a real horse. The motion of the MechaHorse would help build confidence in the user and train core muscles required for horse riding. Therefore, it also acts as a safer training option when the rider is still not fully ready to take on a real horse or when a horse is not readily available.

1.2. Objectives

Our objectives are as follows:

1. Motorize the MechaHorse.
2. Design and install a drive unit to provide motor to crankshaft transmission.
3. Correct the walking motion of the original MechaHorse.
4. Add a 2nd mode of motion: Trotting.
5. Establish an actuator-to-saddle support mechanism to switch between the 2 modes of motion.
6. Control of motor speed through a smartphone app.
7. Addition of a physical stop button to ensure immediate disengagement from motor.
8. Addition of weights to the bottom platform of the MechaHorse to improve stability.

1.3. Background Research

Scientific journals mapping horse motions were studied where we approximated the general motion pattern when horses are in different modes of movement. A rider on the back of a walking horse will experience an approximate circular motion of 30mm radius[2]. If the horse is in trotting motion, the rider will experience a vertical elliptical motion of 30mm x 60mm[3].

[1] http://horseshelpingpeople.co.uk/pages/therapeutic-sessions/how-can-it-help
[2] http://www.sciencedirect.com/science/article/pii/S0167945714001778
[3] http://jeb.biologists.org/content/216/10/1850.figures-only

1.4. User Requirements

a. User height has to be between 120-190cm.
b. User weight has to be between 40 -100kg.
c. User age has to be above 7 year-old.
d. User disability is categorized as mental disability.

1.5. Technical Requirements

1.5.1. Structural Requirements

a. MechaHorse shall have dimensions of 1200mm x 500mm x 1000mm.
b. MechaHorse shall have maximum total weight of 100kg to ensure portability.
c. MechaHorse needs to be stable during usage.
d. MechaHorse needs to be suitable for indoor use in non-humid and dry environment.
e. MechaHorse shall have a life of at least one year between check-ups and maintenance.

1.5.2. Gait Requirements

a. MechaHorse shall be able to provide two motions: walking and trotting.
b. Walking needs to be in a 30mm radius circular motion.
c. Trotting needs to be in a 60mm x 30mm vertical elliptical motion.
d. Saddle movement is in a 2D plane.

1.5.3. Mechanism Requirements

a. Motor shall be powered by electricity.
b. Motor shall be directly transmitted through crankshaft.
c. Motor has to be able to drive crankshaft at 1-3Hz.
d. Motor shall be able to provide a cosinusoidal force of frequency 2π and amplitude 1.2kN in the y and x direction (support minimum load of 130kg).
e. Power System should be able to run in full operation for a period of 8 hours.
f. MechaHorse should be able to switch between the two horse motions, walking and trotting.

1.5.4. Control Requirements

a. MechaHorse gait type and frequency shall be controlled through an app.
b. Circuitry in MechaHorse shall be linked by Arduino.
c. MechaHorse shall be brought to a gentle stop through control mechanism in Arduino.
d. MechaHorse shall have an easily accessible emergency stop button in case of circuit failure.

1.5.5. Safety Requirements

a. Mechanical and drive system should be covered to prevent easy access to machinery.
b. Control system has to be able to bring the motor and the MechaHorse to a controlled stop.

1.6. Cost Requirements

a. The MechaHorse should cost under £2000.

2. Final Design

2.1. Overview

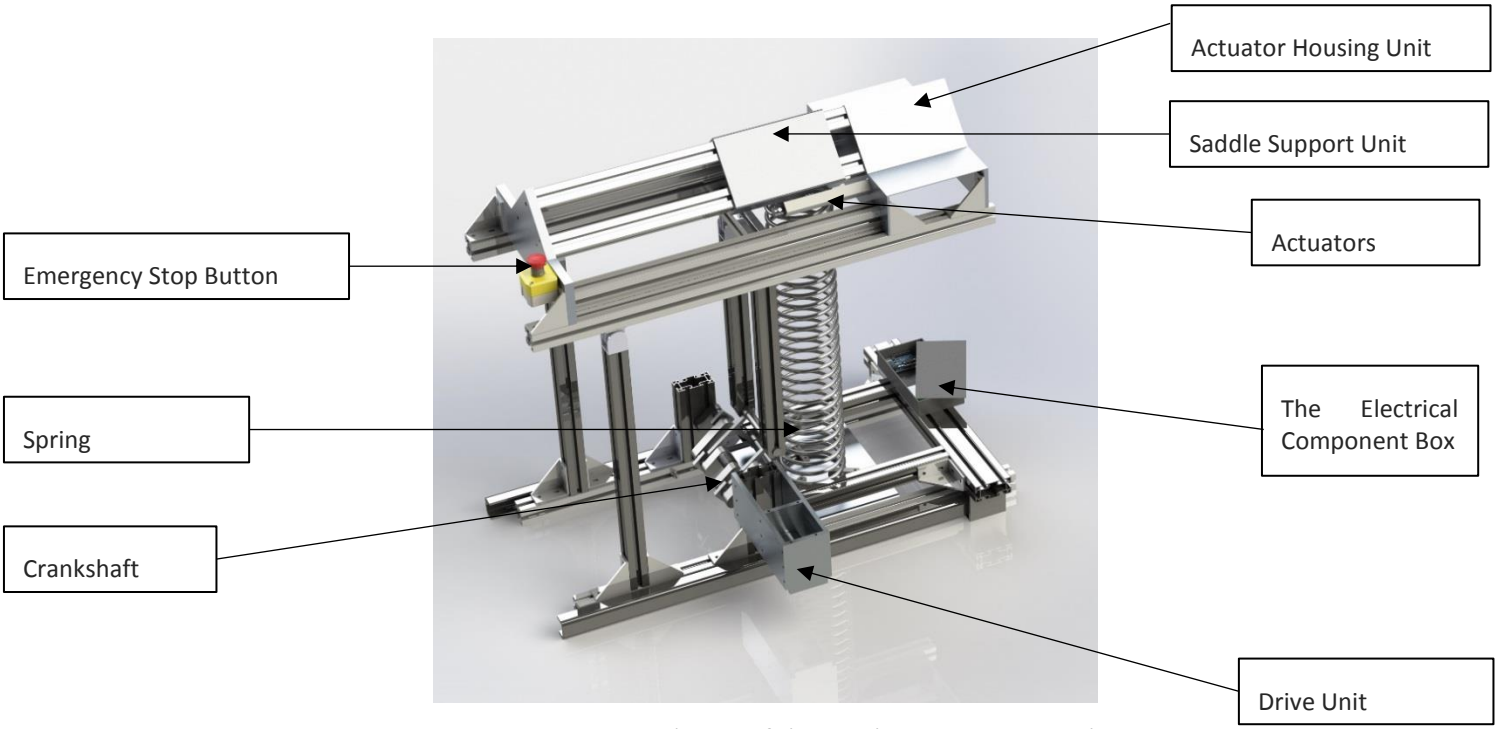


Figure 2-2-1 - CAD design of the MechaHorse in general

The final design of the MechaHorse is powered by a single brushless DC motor directly coupled to the crankshaft, which generates vertical and horizontal displacement that mimics the horse riding motions. Two linear actuators are implemented along the horizontal body frame of the horse, to the saddle support unit, to allow for smooth and convenient switch between the optimal seating positions for the two modes: walking and trotting.

The primary framework of the MechaHorse was inherited from the previous group (its CAD design shown below), where the upper platform is attached to the lower platform with the use of knuckle joints and aluminium Flexlink beams. This results in a seesaw motion of the horse body about the joints.

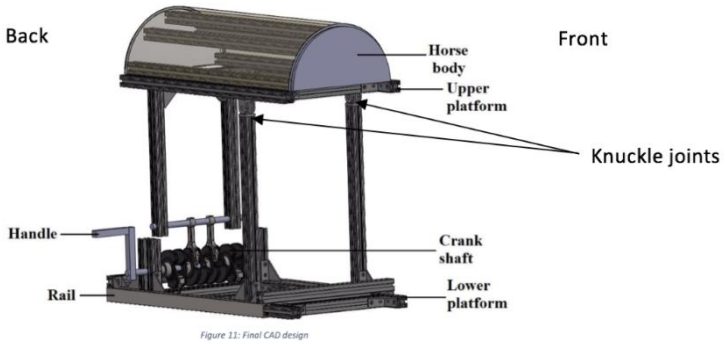


Figure 2-2 - Annotated version of CAD design from previous group's MechoSteed Report

With the use of similar triangles, the vertical displacements of both the walk and trot motion can be achieved. By shifting the 4 pump cylindrical crankshaft to the position X1 (in figure 2.3) and have the saddle directly above it, the vertical displacement for walk is obtained. Likewise, when the saddle is shifted to X2, without a shift in crankshaft position, the trotting motion can be achieved.

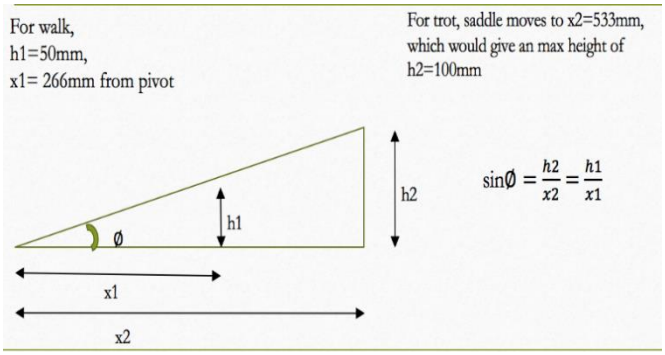


Figure 2-3 - Using geometry to achieve different displacements

To minimize axial loading and shear stress on the crankshaft-motor components, a spring is employed at the rear end of the horse body to offset the weight of the rider. The motor is controlled by the Arduino and its controller, whereby the Arduino is programmed to communicate through Wi-Fi, with a smart phone app, Blynk. Therefore, the user is able to conveniently control the speed, mode, seat positioning, start and stop of the MechaHorse through a smart phone or Tablet.

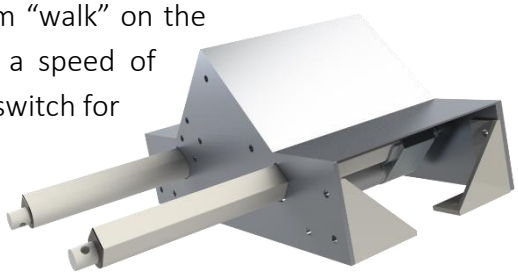
For Safety consideration, a physical Stop/Emergency Button is placed at the balance strap of the saddle. Steel beams are added parallel to the lower platform to lower the overall centre of mass (calculations in Appendix H) and further stabilize the MechaHorse. Stirrups are included to simulate the process of getting up an actual horse as well as to prevent the rider's legs from dangling and getting entangled with the horse mechanism.

2.2. Description of the components

2.2.1. Actuator Housing Unit and Actuators

Figure 2-4 - Link between the actuators and the saddle support unit

The Actuator Housing Unit supports the PA-14 actuators, which can withstand a force of 150lbs, as well as extends the existing horse body. With its end bolted down to a 10mm aluminium plate, the actuator is supported through the hole of the 5mm laser cut aluminium plate. The actuators that have a 12 inch stroke and force are bolted to side of the saddle support unit as shown. When "Trot" is selected from "walk" on the app, the actuators extend steadily at a speed of 0.59inch/s, allowing a safe and gradual switch for the rider.



2.2.2. Saddle Support Unit

Figure 2-5 - Overall operational actuator housing unit

The Saddle Support unit is a 2mm fabricated aluminium sheet that fits nicely above the horse body.

i. To hold the saddle in place

Highly absorbent polyethylene foam is bolted to the metal sheet. When a rider sits on the saddle, the foam absorbs the vertical forces, which dramatically increases its friction coefficient between the saddle and the foam, holding the saddle in place.

Figure 2-6 - Saddle support unit

ii. Reducing Friction

Under the fabricated metal sheet, L shaped steel plates and Polyethylene sheets are bolted together to saddle support unit. Elevation reduces contact area, thereby reducing friction. The use of LDPE Polyethylene sheet reduces friction coefficient from metal-metal of 1 to metal-plastic of only 0.25, further ensuring a smooth transition.

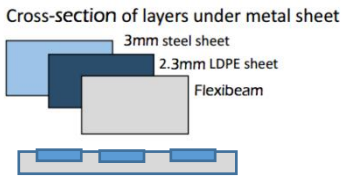


Figure 2-7 - Side view of steel sheet along the Flexlink beams

2.2.3. Crankshaft

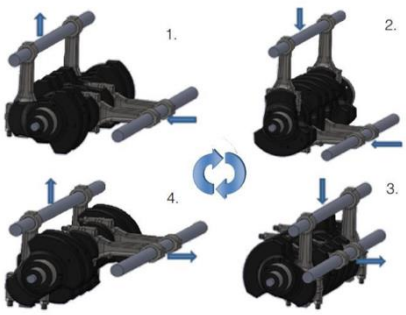


Figure 2-8 - The different stages of an operating crankshaft

The Yamaha XJ600s crankshaft is inherited from the previous group. Flexlink beams are attached to the connecting rods to generate horizontal and vertical motion as the crankshaft rotates.

Since the crankshaft was shifted forward, new stress analysis was done to make sure the connecting rods (parts connecting the crankshaft to the legs) are able to withstand an additional stress (see Appendix I).

2.2.4. Motor

The motor, supplied by Alien Power System, is originally designed for motorizing an E-board[4]. It has a dimension of 63mm diameter and 74mm length, while the shaft is of 10mm diameter wide. It has a power rating of 3.2kW with a maximum current allowance of 120A. Rotation speed is around 2000 rpm, though it should be capable of spinning faster than that.

Figure 2-9 - Motor kit

2.2.5. Drive unit

The motor is held in position within an aluminium box (88 X 88 X 285 mm) aligned with the crankshaft, on the side of the horse. The non-rotating part of the motor is fixed against the box, whereas the rotating shaft coming out of it goes through a pre-designed hole. It provides:

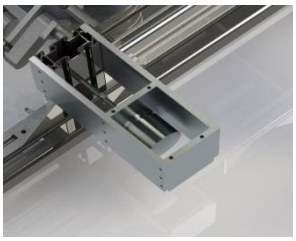


Figure 2-10 - Aluminium Box

i. Protection and isolation of the fast moving parts of the design
ii. A drive unit that follows the horse in its horizontal motion. The motor is coupled directly to the crankshaft via a keyway, making the contact between the already existing motor shaft and crankshaft. The coupling is taken care of by a hollow cylinder bolted on one end to the crankshaft by a fixation pin, and on other end via multiple screws on the rotating part of the motor

2.2.6. Spring assembly

Figu
end plates attached

A chrome silicon spring of 150 mm outer diameter and 10 mm wire thickness, is implemented below the seat position for trot to offset the weight of the rider. The load is estimated at 1.275 kN (30 kg for the horse top and 100kg maximum for the user). Thus, the spring applies a load of 1.2 kN upwards when compressed to approximately 80 cm.

The top coil of the spring is clamped on the top spring plate, just before the actuator housing unit. The bottom coil is clamped on the bottom plate, vertically aligned to the top plate and bolted on the moving frame, such that the spring remains straight and vertical at any time during the motion.

The spring has a free length of 2m and a spring constant of 1089 N/m. The characteristics of the spring were picked [5]and calculated such that the spring would not dramatically change the force it applies as it compresses/relaxes with the motion. Moreover, it can resist to the maximal load (Appendix I).

[4] http://alienpowersystem.com/shop/e-board-kits/aps-120amp-3-2kw-single-motor-e-board-diy-kit/
[5]Juvinall, Robert C, and Kurt M Marshek. Fundamentals Of Machine Component Design. 1st ed. John Wiley & Sons. Print., Chapter 12

2.2.7. Additional safety mechanisms

Emergency Stop button with mushroom type head and twist to reset mechanism. Supply current is 0.55 A DC or 6 A AC.

Figure 2-12 - Emergency Stop button

Stirrups (stirrups: stainless steel, standard size; leathers: 121 cm full length, adjustable) and a balancing strap (16 cm long) to provide stability for the user.

Figure 2-2-13 - Stirrups

Figure 2-14 – Balancing Strap

2.2.8. Electrical components box

The motor controller, the H-bridge and the Arduino are in a plastic (insulating) box at the rear end of the horse, a central position between every electrical component. The box is taped to the rear end of the lower platform beam.

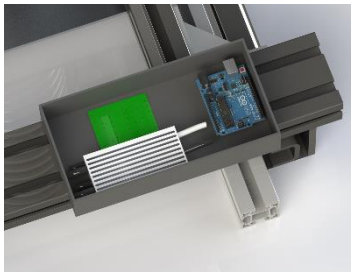


Figure 2-15

Electrical components box

2.2.9. Circuit

The circuit is powered from two 12V batteries which can supply the power for about 30 min of active use before needing to recharge. The commands to the motor controller are sent through the Arduino which we can communicate with through a smartphone.

Final design:

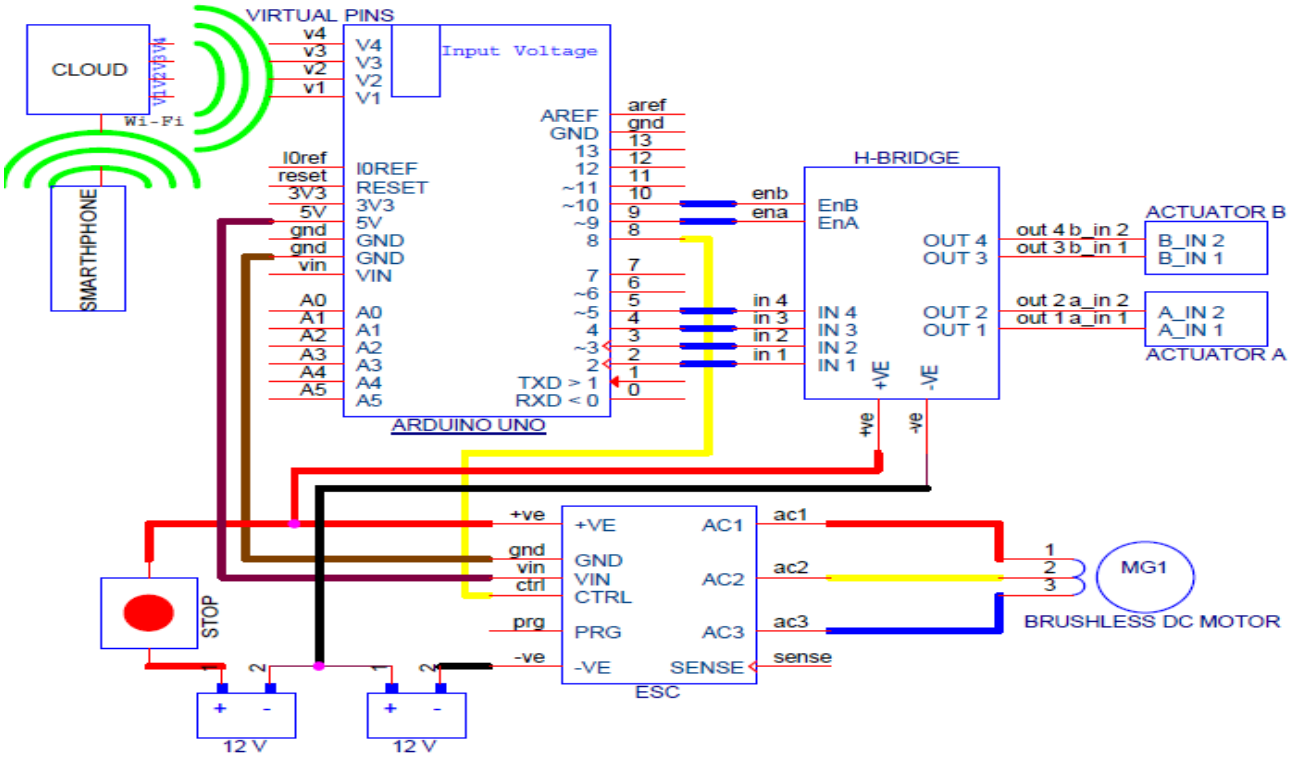


Figure 2-14 Final design

The electric speed controller (ESC) provides three AC currents inputs, that are 120 degrees out of phase, to the brushless DC motor. The ESC is powered by two 12V batteries connected in series. The positive input is wired through the emergency stop button so that the power supply can be cut in case of emergency. The controller requires input of 5V, GND and control input. These are connected from the Arduino with the control input coming from pin 8 which is the one we specified in our program. The Arduino was programed to receive inputs from the smartphone, through the BLYNK app, and send desired values of RPM to the ESC. The BLYNK library allowed us to designate virtual input pins (shown on the diagram) which were programed to receive input from the buttons on the smartphone through

Wi-Fi connection (as opposed to making wired connections). The actuators take +12V input to extend and -12V to retract. They are controlled by the H-bridge which can invert the voltage on command from the Arduino, which itself takes command from smartphone.

Regarding H-bridge, outputs are determined by input signals. Here, IN1 and IN2 are responsible for OUT1 and OUT2, while IN3 and IN4 are responsible for OUT3 and OUT4. To activate the two gates, two separate enabling signals responsible for each gate, EnA and EnB, must be High all the time. Subsequently, if IN1 is High and IN2 is Low, then OUT1 is High and OUT2 is Low, or vice versa.

The HV-BEC cable can diverge a current from battery and provide a steady 6V power supply to Arduino.

2.2.10. Software

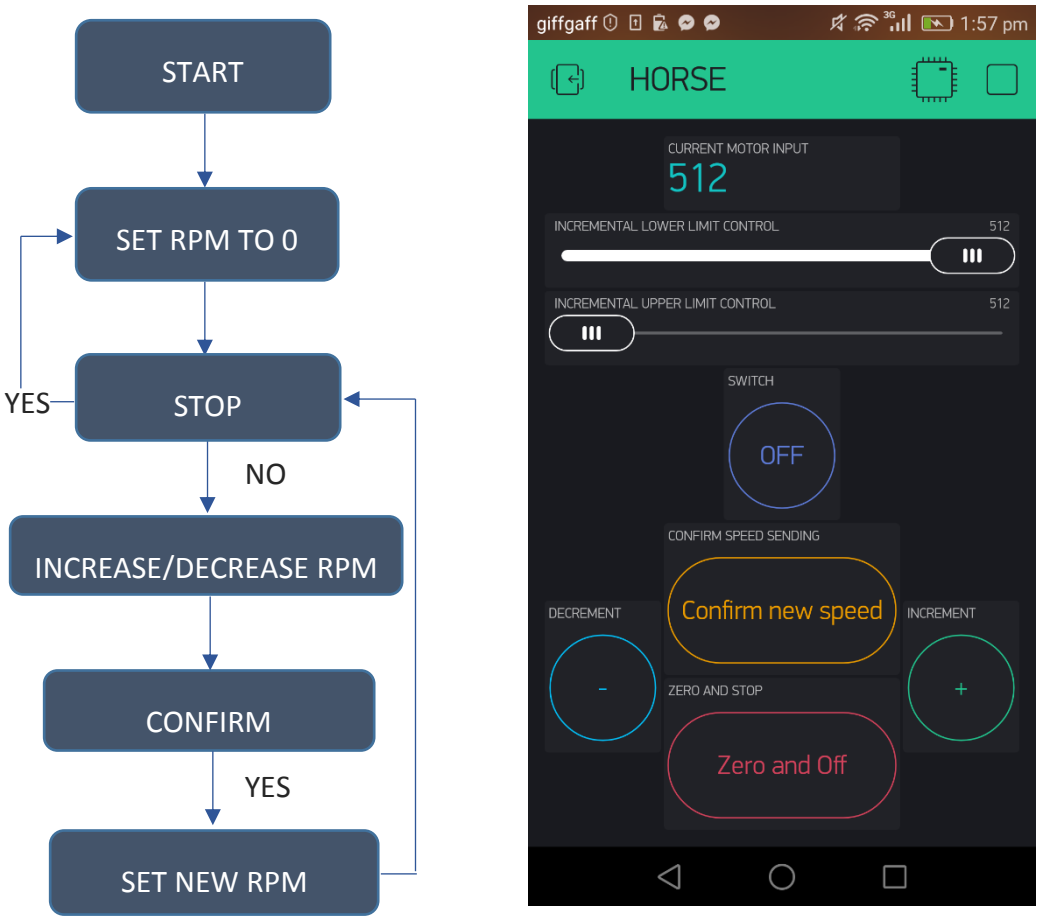


Figure 2-15 UNO/APP Code Flow Chart

The Arduino Code mainly uses Blynk and Servo libraries. The analogue input to the Arduino can take values from 0 to 1023. The Arduino sends values to the ESC from 1000 to 2000 which sets the angle of the motor shaft. On a standard servo motor, a parameter value of 1000 corresponds to fully counter-clockwise, 2000 is fully clockwise and 1500 is the mid-point. At the start of the program the value of 512 (mid-point of analogue input) is mapped to angular value of 1500. We chose this as to allow the same range over which we can control both the clockwise and anticlockwise speed of the motor.

Sending values above 512 will turn the motor clockwise with 1023 corresponding to maximum speed. Values below 512 will turn the motor anticlockwise with 0 corresponding to full speed.

The + and – buttons can be used to increment and decrement the current speed value by 20 respectively. Conformation input must be sent before the program maps the new value and sends it to the ESC. The slide bars at the top can also be used to increase/decrease speed which, within our code, automatically sends the new values to the controller without the need for conformation. OFF button sets the speed value to 512 and automatically sends it to the ESC, which stops the motor.

The APP refresh rate is 1 Hz. This means there is a 1 second delay after the command is sent. See Appendix G for the code and more detailed (technical) explanation.

2.3. Prototype

Ultimately, we achieved the two gait motions with a single crankshaft, implemented a successful housing support unit and saddle support unit that switches the seat position when different modes were selected on the app, as planned. The circuit and App was able to control the motor which resulted in crankshaft motion, under no load conditions. More will be discussed in the next few sections of the report.

The spring was not implemented because the motor could not drive the MechaHorse (more details in Testing and Evaluation). However, the spring assembly was manufactured and put together, and the spring was compressed. As of today, the spring can be put in the horse any time.

Steel beams were not added to the bottom platform of the rod to allow more convenient portability for testing. As there was no test subject above 70kg, the MechaHorse was sufficiently stable without the steel beams. However, they are stored in Princes Garden and can be installed when the MechaHorse is complete. Due to budget constraints, stirrups and other additional accessories were not added as well.

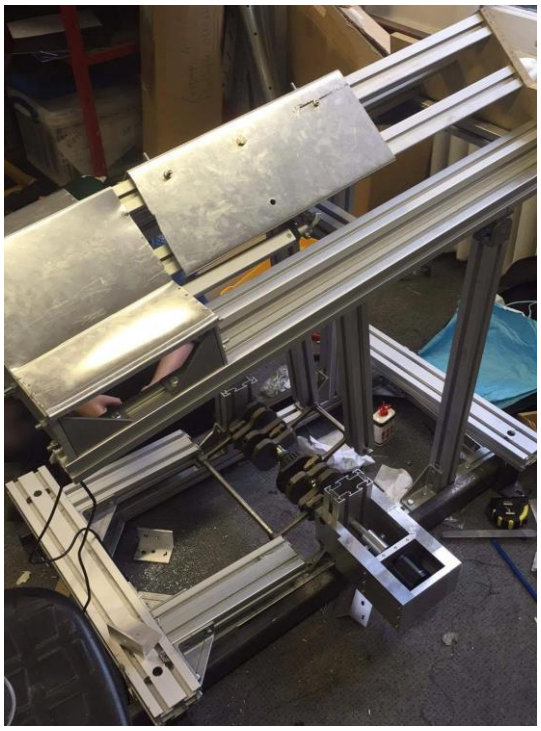


Figure 2-16 - Final Prototype

3. Testing and Evaluation

3.1. Physical testing

3.1.1. Evaluation on drive system

Although every individual component functions properly after assembly, the crankshaft was not able to drive the MechaHorse in loaded conditions. There are the 2 primary reasons for this:

1. The motor is designed for generating high rotations per minute (rpm) motion.

While the power specification stated that the motor can provide up to 3.2 kW, its operating frequency was way beyond what was required, which also meant it could not provide the torque unless the motor was geared down.

The motor was inherited from a previous group who worked on this project in the past where both them and the supplier provided us with no information on this motor, the problem was only discovered after the control mechanism had been recovered, which was too late in the development stage to purchase a new motor.

To visualize the problem more clearly, tests were conducted to measure rpm with respect to different input signal, under no-load condition.

In the test, we attached the motor on a clamp, stuck a reflective tape on the motor, powered it, sent in different values, and then measured the rpm with a stroboscope.

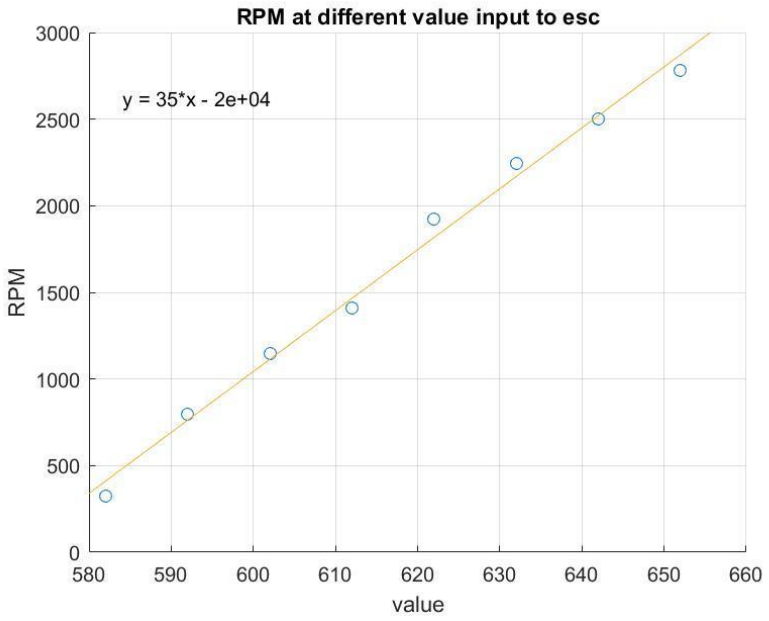


Figure 3-1 Graph of RPM vs value inputs

Value refers to the signal sent to motor, which varies from 0 to 1024, and the zero point is defined at 512. According to our test, it is found that the minimum speed occurs at around 400 rpm. Still, the motor should not be operated at this speed, which limits the torque it provides, making it very susceptible to resistance.

Therefore, operating range is between 2500 to 3000 rpm, which is about 45 – 50 times of the desired 60 rpm.

2. The spring provided too much resistance

The spring was too strong and provided an upward force of around 100kg when compressed. This was meant to cancel the weight of users, but proved too to be inconvenient to implement. Testing was also difficult because this spring-support system would only be effective when there is a user sitting on the MechaHorse. Furthermore, spring support system would prevent any further editing of the assembly, as it would be complicated to disassemble some parts without the spring expanding. Finally, the damping effect of the spring would hinder the downward motion of the MechaHorse body. Hence, the potential benefits of the spring was not maximized and optimal motor conditions could not be achieved.

3.1.2. Evaluation on seat-adjusting mechanism

The actuators adjust the seat as planned. Its effectiveness in switching modes are evaluated in the 'evaluation on new motion' section.

3.1.3. Evaluation on user interface

The control app can properly control motor rotation speed. Though, occasionally the app may lose connection with Arduino. This can be due to random interference from other electronics.

The app interface has speed values that may not be intuitive to the users. Therefore, a user manual was constructed to guide the user (shown in Appendix F).

3.1.4. Evaluation on safety

Safety mechanism is implemented to protect users in case of failure. These includes an emergency stop button, a fuse, and some built-in function within the ESC (Electric Speed Controller). Below is a list of scenarios and corresponding outcomes.

Power supply being cut off	Motor stops abruptly
Physical emergency button being pressed	
Bad connection between ESC and Arduino / signal failure	Motor stops as ESC has a safety mechanism which stop motor in case of signal failure
Control app failure (e.g. app crashing)	Motor will continue running until app is reloaded and stopped properly. Or emergency stop button can be pressed.
Motor stalling	Motor stops as ESC has another safety mechanism which stop motor in case of stalling
Short-circuit (I.e. when I > 150A)	Motor stops as the fuse is burnt

As the motor could not drive the MechaHorse when implemented, such scenarios could not be tested for. However, when the motor stops, the MechaHorse should stop being powered and would likely decrease speed and come to an eventual stop.

3.2. Computational Analysis

3.2.1. Evaluation on new motion

The ideal trotting motion had a 30mm x-displacement and 60mm y-displacement, and the ideal walking motion with 30mm x-displacement and 30mm y-displacement. The accuracy of this motion was tested by filming a video of a rider on the horse, and following her motion using Kinovea software, comparing the acquired data(blue data points) to the average requirements(red line) afterwards (Figure 4 and 5). This showed that the average characteristic y-amplitude for both walking and trotting was almost exactly achieved, alongside a similar x-displacement, however this was inconclusive because the MechaHorse was manually driven during this test, resulting in an unsmooth motion, as well as a slight shift forward in the x-direction. As the horizontal displacement was already achieved by the original framework, it can be reasonably concluded, that the two vertical motions were effectively implemented. Additionally, the switch between motions was successfully controlled by the linear actuators.

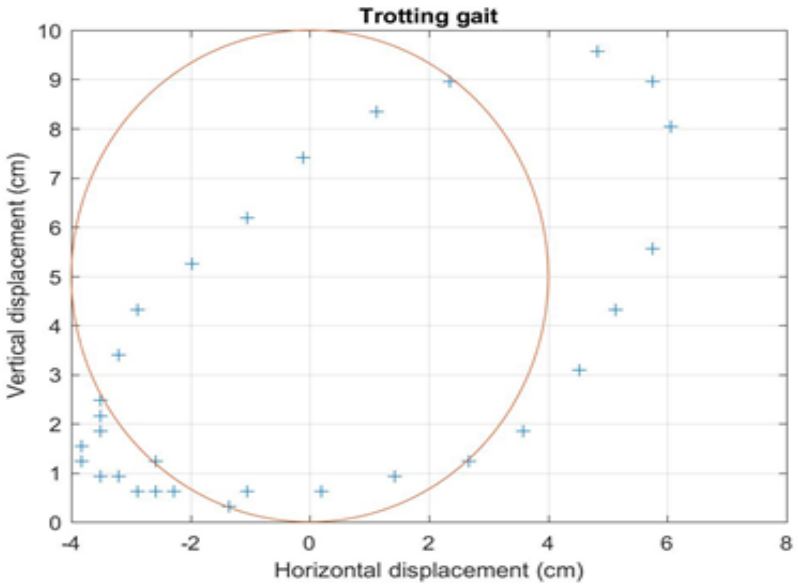


Figure 3-2 Plot of vertical and horizontal displacement for trotting gait

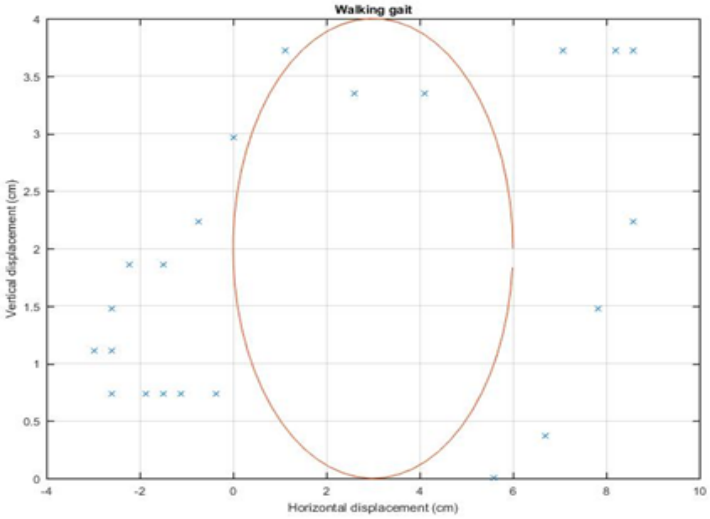


Figure 3-3 Plot of vertical and horizontal displacement for walking gait

3.2.2. Evaluation Matrix

Correct the motion	✓
Implement the motor	✓
Implement the spring	✗
Implement a seat-adjusting mechanism (for switching between walking and trotting mode)	✓
Motorize the crankshaft	✓
Motorize the MechaHorse	✗
Devise a control mechanism over the motor	✓
Implement a control interface	✓
Implement an emergency stop system	✓

4. Discussion

For the MechaHorse 2.0, we sought to build upon the inherited framework to (i) correct flaws in the mechanical design and (ii) motorize it. For (i), the main flaw in the design was the gait of the horse. We successfully provided two types of motion—walking and trotting—whereas the previous group had only developed one and we achieved a gait that was much closer to the desired gait than previously. For (ii), while we were not able to fully implement our electrical design into the MechaHorse framework, the electrical and software components of the design did work well in isolation. Our major constraint was our budget which was not able to cover the breadth of our scope. It greatly hindered our capacity to find solutions to our concerns and limited our control over the project. Overall, the individual facets of the design worked fine on their own but we faced major hurdles in their integration.

Mechanical Design

The original mechanical design faced a number of issues that we sought to correct. The gait was incorrect and did not achieve the y-displacement we wanted. We were able to correct this by shifting the crankshaft forward and alternating the user between positions corresponding to the type of motion. We were also able to implement a means to switch between the two gaits—which requires moving the seat of the user—electronically and while the user is still seated. Had our motor been compatible with the design, this would have been a seamless user experience.

In theory, the spring would have offset some of the weight of the user. However, in practice, as we shifted the crankshaft, there wasn't much room for the spring to move. We thus decided against implementing the spring,

Electrical and App design

The software part of the project revolves around the use of Blynk, a licensed application available on both Google Play and the App store.

More precisely, the core software involves an integration of:

(i) The program required to run and synchronize Blynk on the cloud
(ii) The program required to control a Servo Motor, adapted to control our brushless DC motor
(iii) The program required to move the actuators in their walk or trot positions

The programs are all run and compiled on the Arduino IDE. The external libraries included are:

(iv) Simpletimer.h
(v) Servo.h
(vi) Blynk.h
(vii) SoftwareSerial.h
(viii) BlynkSimpleStream.h

The first two programs are integrated and work seamlessly alongside each other. This allows for very intuitive control of the motor speed thanks to the obviously displayed buttons. However, the third program has to be run and uploaded separately as Blynk uses a serial reading rate of 9600 bauds, which is the exact same for that of the actuator control: values are sent through the serial monitor at a rate of 9600 bauds and running both at the same time interferes with the proper working of the Blynk cloud synchronization. This issue unfortunately means that changing of the Mechahorse gait

has to be done on the Arduino Serial monitor on the computer, as opposed to the rest of the control done on the App.

The ideal case is thus when all three programs are integrated to work as a whole, essentially when speed control as well as gait switching take place on the app interface in real time without interference. This potential improvement is currently unachievable, however, as there is no known way to work with two baud rates at once; thus we cannot pair communication throughout the serial monitor and the Blynk cloud.

The programs are included in the appendices and are commented to allow for better comprehension.

Future solutions

For added safety, the motor drive unit should be covered with another piece of aluminium to protect the unit and prevent tampering of the unit. This simultaneously can act as a step for the user to mount the horse. For the App interface, a timer showing 'Duration Elapsed' could be added so the user knows how long they've been using the horse. Additionally, the linear actuators should also be included in the app interface such that it can be controlled via the smartphone directly. Stirrups can also be added for better simulation of horse-riding.

Our main area for improvement would definitely be the motor drive unit.

We could solve its outstanding issues by introducing a gear box of approximate ratio 1:50. Since space is limited, the gearing ratio is probably achieved by introducing pairs of smaller gears instead, which also means there will be higher energy loss due to friction. Alternatively we could introduce a new motor, which can provide high torque, low rpm. Such motor should be able to rotate at 60 - 180 rpm with at least 1.5 horse power (1119 Watt). However, such a motor can be very expensive and hard to find.

The solution we would be putting forward would be a combination of the above 2 options.

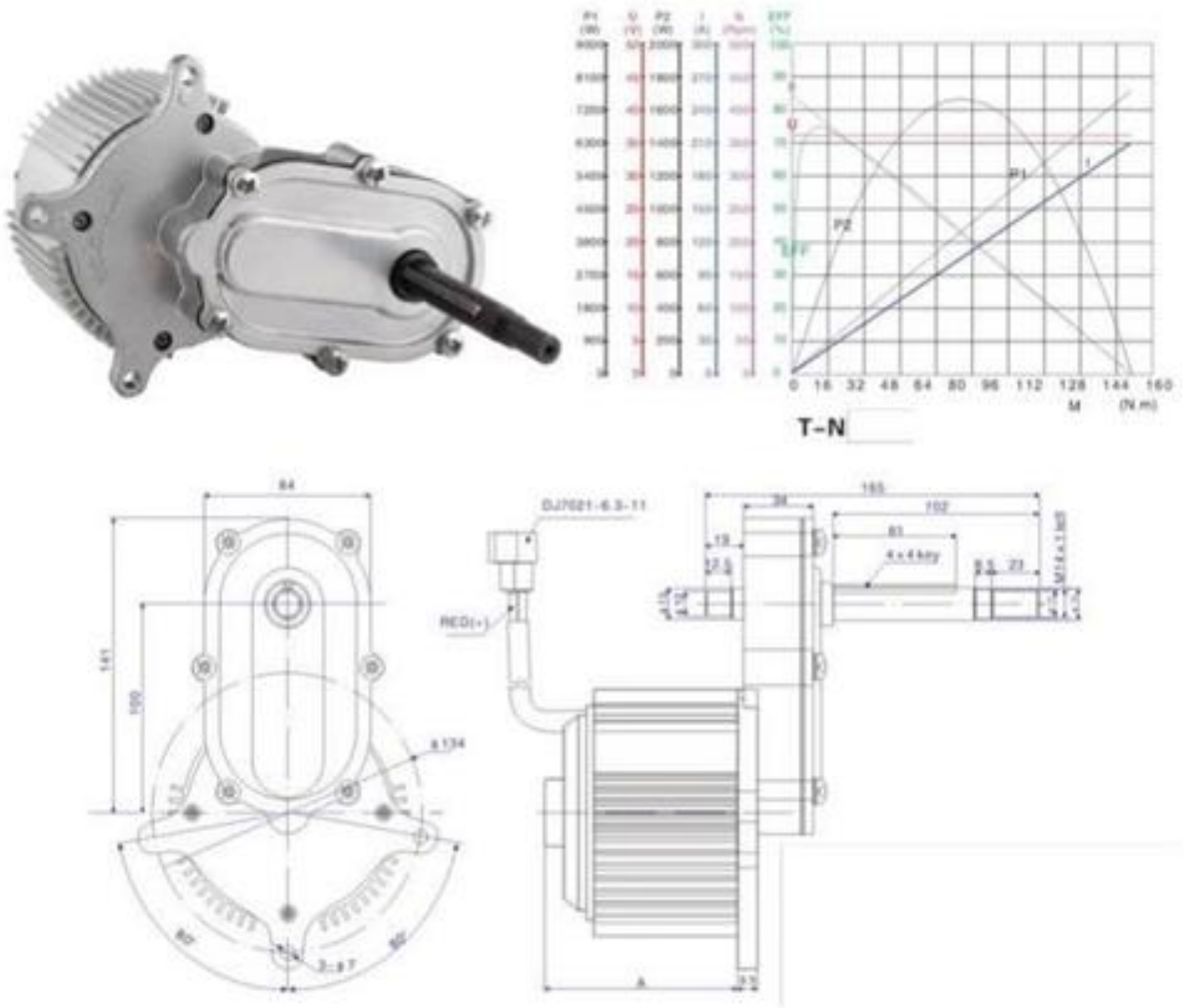
Control interface can be another area for improvement. It can be modified in a way that the user can directly modify the speed or a scale of speed rather than 'value', a comparatively vague parameter, so that they can visualize the change more easily. Another possibility is to fix the value sent to motor and introduce discrete 'modes'. For example, if user chooses the mode 'slow walk', then the motor will rotate slowly at speed ω at position A.

Currently, our motor has a high frequency output but should be replaced with one with a low frequency output. A possible motor model is XYD-18a, which can provide 80Nm in average and rotate at 320 rpm. In fact, another EDP group working on a stair-climbing wheelchair has one of this spare. If this project is picked up by any future group, hopefully it is still there. Or else, it cost around 50-100 USD, which is affordable by EDP funding. If such motor is to be used, it would need to further gear down by 1:4, which is much more manageable.

Details of motor are in Figure 4-1 on the next page.

"The wheelchair's drivetrain will consist of four XYD-18a 1kW Brushed DC Motors. These motors feature a stall torque of 168Nm, and a no-load speed of 420 RPM." - Interim Report

Each of the four driven wheels is driven by its own motor which is mounted to the wheel's upright.



Motor characteristics and technical drawing

Each motor's axle is attached to a worm gear, which meshes with a worm wheel attached to wheel's axle, forming a gear reduction to provide the necessary torque to complete the stair obstacle of the race. Both the worm gears and worm wheels are provided by RS Components.

The rear wheels have a 50:1 gear reduction, which is formed from the 521-6890 and 521-6979. The front wheels have a 15:1 reduction, formed from the 521-6941 and 521-6907. The specifics of the gears used can be found in APPENDIX I.2.

Each wheel also has a freewheel system operated by an Arduino controlled, stepper motor actuated, screw jack. The assembly is attached between the wheel upright and motor mounting brackets. The pilot flips a switch to toggle between the drivetrain's engaged and disengaged states

The motors are wired to two RoboClaw HV 2x60A motor controllers, with the rear motors and

Figure 4-1 Details of proposed motor

5. Conclusion

In conclusion, this report properly illustrates all the steps taken in order to achieve our end goal of creating a therapeutical equine simulator.

Both gaits: walking and trotting, were deemed achieved thanks to verification of the motions generated through movement via the Kinovea software. A circular and elliptical motion respectively were successfully recorded.

Successful modifications to the positioning of the crankshaft and creation of a supporting block were achieved through design and prototyping in SolidWorks. Numerous calculations were ran in order to minimize chances of failure. The positioning of the crankshaft as a whole was drastically modified as to allow for both gaits to be achieved: past design of the MechaHorse only allowed for one slightly flawed walking motion.

Movement of the crankshaft originates from the motor, which is in turn controlled through an external App named Blynk. The core of the Blynk programme is set up through the Arduino IDE while the control interface is configured on the app itself and is intuitive of use. It is to be noted that the main software used to control the motor is a servo motor control program adapted to work with our brushless DC motor. Mode switching is done through a program, which changes actuators position. Because of certain software related obstacles, the two programs were not fully integrated but work perfectly on their own.

Unfortunately, the final goal of fully motorizing the crankshaft was not achieved. This is in part because we were not provided with a torque to RPM sheet despite asking the provider – thus leaving it entirely to trial to see if it was indeed possible to move the crankshaft attached to the main frame of the MechaHorse with the provided motor. We sadly fell short, though full control of the crankshaft on its own was achieved, as shown by video footage.

The spring currently does not allow for testing without users due to its intended function of applying 100 kg upwards. It is however compressed and ready to insert into the frame of the MechaHorse at any time.

In conclusion: most of the goals established early on were accomplished, while only the final few steps were hindered due to budget and time constraints. With some additional information regarding the motor, steps could have been taken to avoid the greatest issue of low torque. While not a perfect project, this was certainly not an unsuccessful one.

6. References

1. Horses Helping People. 2010. *How could spending time at Horses Helping People help you?*. [ONLINE] Available at: http://horseshelpingpeople.co.uk/pages/therapeutic-sessions/how-can-it-help. [Accessed 4 June 2017].
2. Brian A. Garner and B. Rhett Rigby. 2014. *Science Direct*. [ONLINE] Available at: http://www.sciencedirect.com/science/article/pii/S0167945714001778. [Accessed 4 June 2017].
3. Patricia de Cocq, Mees Muller, Hilary M. Clayton, Johan L. van Leeuwen. 2017. *Journal of Experimental Biology*. [ONLINE] Available at: http://jeb.biologists.org/content/216/10/1850.figures-only. [Accessed 4 June 2017].
4. Alien Power System. 2017. *APS 120Amp 3.2KW single motor E-board DIY Kit*. [ONLINE] Available at: http://alienpowersystem.com/shop/e-board-kits/aps-120amp-3-2kw-single-motor-e-board-diy-kit/. [Accessed 4 June 2017].
5. Juvinall, Robert C, and Kurt M Marshek. Fundamentals Of Machine Component Design. 1st ed. John Wiley & Sons. Print.

Appendix A – Project Management

Project Manager

Chun Lin Yap

Finance Manager

Timothee Gathmann

Resource Manager

Emma Duncanova

Mechanical Design and Implementation

Timothee Gathmann

Piramol Krishnan

Magdalene Ho

Jill Ueng

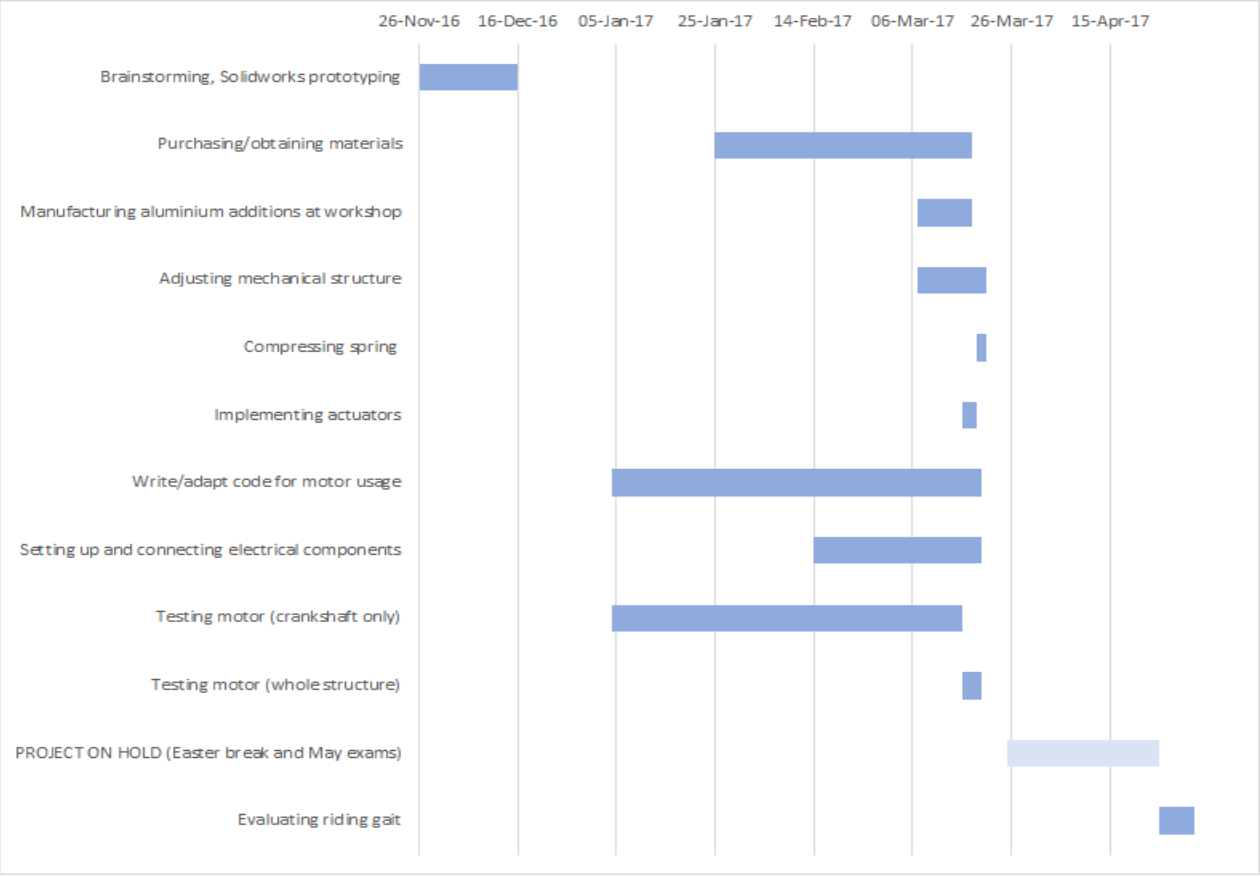
Electrical/Software Design and Implementation

Matthew Kwan Kiu Chan

Luka Lagator

Yann Zhong

Gantt chart



Appendix B – Risk Management

EVENT	HAZARDOUS EFFECT(S)	SEVERITY	CORRECTIVE/MITIGATING MEASURE(S)	LIKELIHOOD (WITH PROTECTIVE MEASURES)
IMPROPER HANDLING OF BATTERIES				
Bending forward to lift batteries from lower ground	Back injuries, unstable posture when lifting, may drop batteries on self	3-4	Go through correct lifting techniques for batteries with trained individual, make sure trolley is braked	1
Poor support for batteries, such as only using one arm or hand to carry them	Wrist injuries, may drop batteries on self or on hit others	2-4	Must not transport by carrying, use trolleys at all times, walk in groups of three (two people supporting trolley, one person supervising)	0
Unstable posture such as twisting body while lifting	May drop batteries on self or hit others, may cause some injuries	2-4	Must not transport by carrying, use trolleys at all times, walk in groups of three (two people supporting trolley, one person supervising)	0
Not paying attention to environment, may trip/bump into something while transporting batteries	May drop batteries on self or hit others, may cause some injuries	2-4	Must not transport by carrying, use trolleys at all times, walk in groups of three (two people supporting trolley, one person supervising)	2
CONNECTING BATTERIES TO CIRCUIT				
Connecting batteries to circuit without a component with high enough resistance between battery terminals)	Short circuit, electric fire, battery explosion	2-5	Circuit must include a fuse in the correct position, group members must be wearing protective eye and body clothing, must build circuit correctly	1-2

Touching the batteries and another shorting bar such as any metal component		3-5	Unneeded metal components must not be within reach, must be aware of surroundings and actions at all times	2
TESTING/RUNNING SIMULATION				
Incorrectly set up circuit (refer to above category)	Short circuit, electric fire, battery explosion	2-5	Completed circuit must be approved by Dept. Electronics Engineer, must not have naked wires (also see above)	1
faulty wiring between mechanical structure, motor, and batteries (due to physical disturbance occurring after the circuit has been wired for the first time)	Short circuit, electric fire, battery explosion	2-5	No conductive elements exposed, batteries are covered with some hard, protective material to prevent users from touching them/damaging the circuit	1
HAZARDS DUE TO MECHANICAL STRUCTURE				
Mechanical workshop manufacturing (folding plates, making clamps, cutting beams, drilling holes)	Injury from using equipment and machinery carelessly or incorrectly	2-5	Must wear protective clothing and eyewear, amber-coded machines must only be used under supervision from trained professionals	1
Spring compression	Injury should the spring uncompress unexpectedly	3-5	Must be done in a workshop environment under the supervision of trained professional, using the proper machinery	2
Implementing structural changes (adjusting vertical and horizontal beams on existing horse structure and attaching	Injury to riders or group members should the structure derail, or should beams shift out of	1-3	No group member can work alone, always under the supervision of others, every change must be confirmed to be stable. Stress analysis was done beforehand in SolidWorks.	0

manufactured aluminium additions)	place due to not fixing them tightly			
OTHER				
Passive battery storage	Battery acid spill, may corrode simulator structure or skin	1-3	Regularly inspect battery seals for signs of corrosion, store batteries securely to prevent accidental contact. Store batteries in the case designed for them.	2
Battery charging	Battery acid spill, electric shock, electric fire	1-5	Charging must be done under the supervision of a "competent person" and in a laboratory setting with adequate ventilation, low risk of fire, and fire protection equipment nearby, and battery cannot be overcharged Fire/explosion prevention: the correct order when charging is mains to charger to battery; then to disconnect, disconnect the mains from the charger then the battery from the charger	2

LIKELIHOOD	SEVERITY
0 = no chance of event (0%) 1 = low chance of event (<10%) 2 = may occur (~50%) 3 = will occur no matter what (100%)	1 = cosmetic defect having little bearing on the effective use or the operation of the device 2 = slight discomfort, or maintenance required on system before attempting next usage 3 = minor injury to rider/supervisor and/or device malfunction 4 = potential of causing major injury to rider/supervisor and/or device malfunction 5 = potential to cause death and/or major system failure

Appendix C – Manufacturing

RSM Workshop: Drive Unit and Motor Box

A thick aluminium box housing the motor, connected to the drive unit and controller on one end and the crankshaft at the other end.
Manufactured by Mr Nardini at the Royal School of Mines workshop.

Mechanical Engineering Workshop:

Saddle Support Unit:

A 2mm aluminium sheet of length 300mm was fabricated to fit the horse body. Holes were drilled into the sides to bolt the actuators to it. L shaped steel plates and LDPE Polyethylene sheets were sawed to length of 8mm and bolted together to the saddle support unit.

House/Back Extension Unit (Actuator Housing Unit)

The 5mm laser cut and deburred aluminium plate supports the actuators. The 2nd aluminium plate that actuators are bolted to were band sawed and deburred to its desired size. Holes were drilled with a handheld driller into the lateral edges of the plate. The top cover of the housing unit was fabricated with holes drilled to its edges. The top cover was then aligned with the aluminium plates and riveted in place.

Figure C-1: Side and Diagonal view of the Housing Unit on the MechaHorse

Spring Clamps

Aluminium blocks were band sawed into its desired length. 8mm Holes were drilled on both sides because the block was too thick to be drilled through. The blocks were then band sawed symmetrically through the holes to convert these into clamps.

Spring Plates

Aluminium alloy 6063 plates were ordered, band sawed, drilled into and fabricated. The spring was bolted to the plates and held in place with the clamps.

Beam Modification

As the crankshaft position is shifted forward, the fixed beams connected to the connecting rods had to be remanufactured. Therefore, FlexLinks were cut to the desired length, shaves and deburred to a curve at the end. 8mm holes were drilled to allow it to be connected to the crankshaft.

Appendix D – Bill of Materials

Bill of materials: Electrical components

Name of component		Description		Quantity	Price (per object)
Fuse		Current allowance : 150A		2	£9
Fuse holder				1	£18.74
Power Switch		Switch by key. Current allowance : 150A		1	£8
H-bridge chip		Consists of 2 H-bridge, each of which has a current allowance of 5A		1	£3.42
Banana plug into ESC		Made of copper		5	Provided by supplier
Emergency stop button		Model: Schneider Electric Harmony E-Stop, Mushroom Head, Supply Current: 0.55 A (DC) , 6 A (AC), Reset method: Twist to reset, Push button actuation: Trigger action, Supply voltage: 600V (AC/DC)		1	£16.11
Motor kit	ESC controller	Provider: Alien power system	Battery voltage 2S~12S Dimensions: 80 x 50x 20mm Mass: 200g	1	In possession
	Motor		Brushless 3.2 kW Dimensions: Body: 74 x 63mm Shaft: 10mm Mass: 750g	1	In possession
Arduino Uno				1	In possession
Copper wires		Copper wires (6 AWG, 12 AWG, 28 AWG)		2m	

Bill of materials: Structural components

Note: This will not detail all of the individual structural components of the MechaHorse which were fixed by the previous project group. All aluminium plates originate from Aluminium Warehouse.

Name of component	Specifications	Quantity	Price (per object)
Bottom spring plate	Material: Aluminium alloy 6063 Dimensions (mm): Length:440±1 Width: 280±5 Height: 54±3 Thickness of plate:10(+5)	1	£4.15
Upper spring plate	Dimensions (mm): Length: 243 Width: 160 Thickness: 10(+5)	1	£4.15
Plate with actuator holes	See appendix	1	£7.35 Laser cutting estimate:£45
Plate without actuator holes	See appendix	1	Materials in possession
Motor unit	See appendix	1	Materials in possession
Foam		~1 m^2	In possession
LDPE Polyethylene		~0.5 m^2	In possession

1

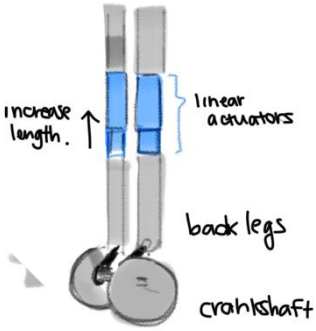
Bill of materials: Mechanical components

Name of component	Specifications	Quantity	Price (per object)
Spring	See appendix	1	£160
Linear Actuator	Stroke: 10 in Force: 15.87 kg - > 155.5 N	2	~£86.70 ($108.99)
Keyway		1	Materials in possession

Appendix E – Initial Ideas and Design

Initial Ideas for Achieving 2 Gait Motions

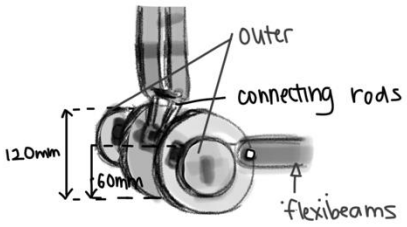
1. Linear Actuator



Linear actuators would have been the most convenient and effective solution to achieving both the walk and trot motion. By adding the linear actuators to the back "legs" of the horse, it can be programmed to produce an additional vertical displacement to the already existing vertical motion generated by the crankshaft. This would allow the trotting motion to be achieved. If the linear actuators are not powered, the "walking" gait is simulated by the vertical displacement of the crankshaft alone. This method would have resulted in minimal changes to the existing MechaHorse mechanism as well as allow the rider to remain in the same position for both positions.

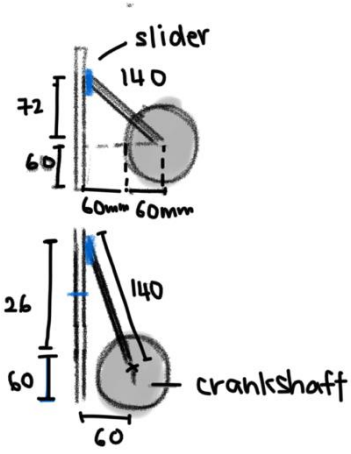
However, the linear actuator would have to accomplish a total displacement of 60mm up and down, with a frequency of 2Hz. Due to the constraints of the budget, the linear actuators that could match the speed and the force needed, could not be obtained.

2. Bigger Crankshaft



Ideally, a 4 pump cylindrical crankshaft with outer radius of 60mm and inner radius of 30mm, would achieve the trotting motion. Using geometry as demonstrated in the report, a change in seat forward, position will allow a walking motion. The crankshaft can remain at the rear end, directly below the trotting position, thereby having a smaller moment to overcome. However, such a crankshaft could not be found in the market as they're usually customized.

Hence, the idea of an overall bigger crankshaft with radius 60mm was explored. With the same mechanism and position as the abovementioned crankshaft (at the rear end, below trotting saddle position), the horizontal displacement caused by the crankshaft can be reduced via a slider mechanism. However, an affordable crankshaft of these specifications could not be found.



Improving Stability and Stability Calculations

Proposed plan: Add 4 steel beams to the bottom platform of horse, 2 to each side (as shown in pink). The extra weight added to the bottom platform of the horse, will lower the Centre of Mass and increasing its base width, thereby providing extra stability.

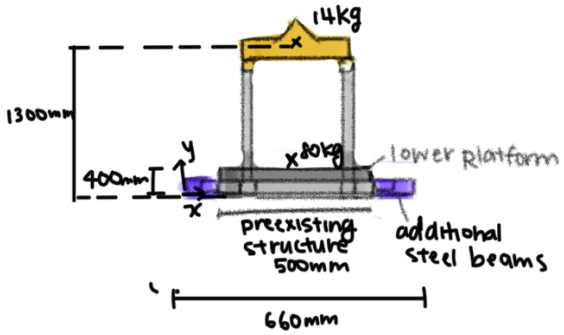
Item	Approximate Mass/kg
Crankshaft	15
Motor	2
Spring	7
6 Steel Bar of 1050mm	38
Connecting Rods	5
Lower Platform	5
Upper Framework	6
Saddle + Housing unit + Saddle Support Unit	8

Total Mass of Upper Horse= Mass saddle + Mass upper framework = 14kg

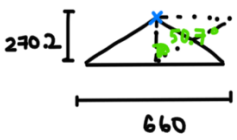
Total Mass of Lower Horse= Crankshaft + Motor + Platform + Spring + Gears + 6 Steel Bars + connecting rods + batteries = 80kg

Calculations 1: MechaHorse structure without rider.

Stability: Very Stable



$$CoM = \frac{80(400)+14(1300)}{14+80} = 270.2mm$$



As its CoM is located 270mm from the ground with a width of 660mm. The horse has to tilt 50.7 degrees before enough moment would be produced to cause it to topple over to the side.

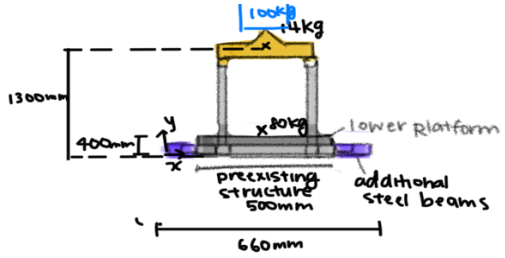
Calculations 2: With Rider sitting on the MechaHorse.

Stability: Very Stable

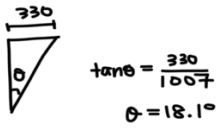
Assumption:

Taking the rider and MechaHorse as a whole structure.

Rider of 100kg has his CoM located near his torso(overestimate, would usually be lower as riders of 100kg would have legs which will move their Centre of Gravity downwards.



$$CoM = \frac{100(1700)+94(270.2)}{194} = 1007mm$$



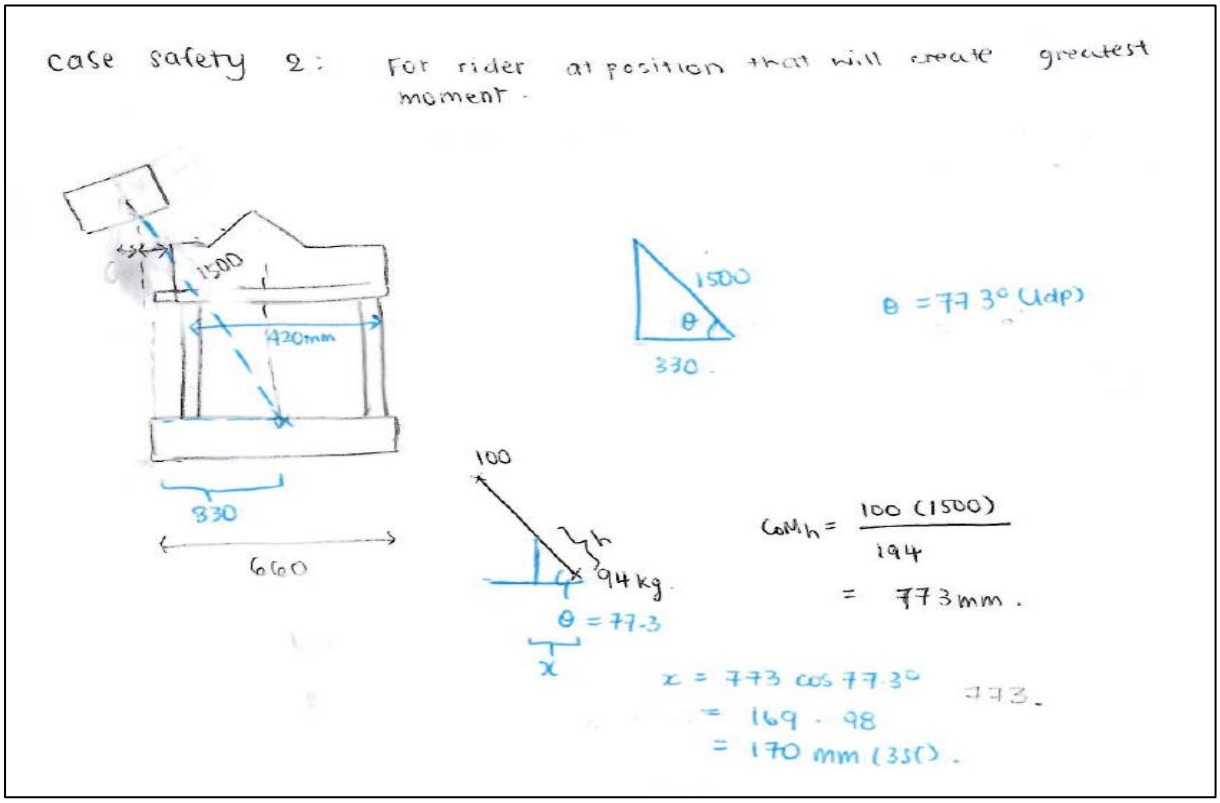
$$\tan\theta = \frac{330}{1007}$$

$$\theta = 18.1^{\circ}$$

The whole horse and rider would have to tilt (as a structure) by 18.1 degrees to its side to cause the whole horse to topple. It would very unlikely happen because if someone were to push the horse (which is already 94kg), a rider's natural instinct would be to tilt the other way to produce a moment that would prevent the whole horse tilting. Therefore, the horse is very stable when the rider is on it.

Calculations 3: When rider is climbing onto horse.

When the rider tries to climb onto the horse, his mass would be result in a new CoM of 773mm (h) along the line from the CoM of horse to the rider's CoM. This corresponds to a distance of 170mm from the x Central axis of the horse which is still within the width/2 of the horse. Hence the horse will not topple.



Also, as the horse body is 120mm inside the outer beams, moments would not be produced on the left of the horse, as the rider tries to climb on the horse.

All the calculations above are done for a height of the horse greater than 1300mm but in fact, the horse is only 1070mm from top to bottom. This gives us plenty of leeway for the CoM of the rider being higher than expected as our riders who are physically disabled may have varying CoM height. This ensures that the CoM of the total horse and rider in reality is lower (towards the ground) and hence more stable than shown in the calculations, which makes even more safe.

Circuit

The initial design was to power the circuit from the mains which would allow us to run the simulator indefinitely. The coupling of the motor was through a gearbox that would allow to transfer the torque of high RPM motor to the horse that was required to move at 1 to 3 Hz. A laser sensor would measure if the motor is going at desired RPM and send feedback to the Arduino. The circuit would supply the power and control to dynamic actuators which would allow a trotting motion.

Appendix F – User Manual

Current motor input shows the value sent and mapped to the motor and this value is refreshed every second (refresh rate of 1 Hz). Notable values are:

- 512: 0 position, noting that this is the default zero position every time the app is started.
- 512 + 80 = 592: clockwise walk
- 512 – 80 = 432: anticlockwise walk
- 512 + 120 = 632: clockwise trot
- 512 – 120 = 392: anticlockwise trot

Incremental lower and upper limit control accomplish two tasks:

1. Update visually with **current motor input** value - if **current motor input** value is below 512, the **lower limit control** bar moves. If **current motor input** value is above 512, the **upper limit control** bar moves.
2. Rather than using the **increment** and **decrement** buttons for more accurate control, the bars can also be dragged to reflect instant change in motor input value. It is to be noted that the bars are coded to work with a built in **confirm speed sending** – meaning that dragging the bars to a new motor value will change the speed of the motor without the need for confirmation. Hence, this is more for testing purpose as it is generally deemed unsafe.

Switch toggles the switch from walking to trotting positions – this button is the only one not integrated in the final app design – reasons for this are explained in the app discussion. The ideal scenario is to be able to detect which gait the horse is currently on and switch to the other one immediately upon a press of the button.

Confirm speed sending takes in the current value stored in **current motor input** and applies it to the motor. For example, if the previously stored value is 512 (default zero position) and the **current motor input** value is 592, the motor will go from stationary to walking speed.

Increment and **decrement** will add or remove 20 to the **current motor input** value. Using of these buttons require pairing with the **confirm speed sending** button.

Zero and stop will accomplish two tasks:

1. Stop the motor if it running by setting the updated value to 512 – the zero position.
2. Reset the **current motor input** value to 512 – faster than using **increment** and **decrement.**

Appendix G – Arduino Code

Motor Control Code

Verify

Motor_mechahorseV3_new

```
#include <SimpleTimer.h>
#include <Servo.h>
#include <Blynk.h>
#include <SoftwareSerial.h>
SoftwareSerial DebugSerial(2, 3); // RX, TX

#define BLYNK_PRINT DebugSerial
#include <BlynkSimpleStream.h>

char auth[] = "a37dcdda959e405b95175a733d22f250";               // Authorization Token for the Blynk App

BLYNK_CONNECTED() {                                             // Whatever is in this will run every time Blynk connection establishes
  Blynk.syncVirtual(V4);                                        // Refresh the value of V4 which is storing the motor value (int val)
  Blynk.syncAll();                                              // Syncs all virtual pins
}

Servo esc;
int val = 512;                                                  // Stores analog motor speed value;
int spd;                                                        // Mapped value
SimpleTimer timer;                                              // Internal Timer

// The function below sends Arduino's run time every second to Virtual Pin 5

void myTimerEvent()
{
  Blynk.virtualWrite(V5, millis() / 1000);                       // Reading frequency occupied by VIRTUAL PIN 5
  if (val <= 512) {
    Blynk.virtualWrite(V6, val);                                 // Slide bars updated with motor speed value every second
  }
  if (val >= 512) {
    Blynk.virtualWrite(V7, val);
  }
}

void setup()
{
  // Debug console
  DebugSerial.begin(9600);                                       // CAN'T OPEN SERIAL MONITOR WITH BLYNK AT THE SAME TIME

  Serial.begin(9600);
  Blynk.begin(Serial, auth);

  Blynk.syncAll();                                               // Syncs all virtual pins

  // Setup a function to be called every second
  timer.setInterval(1000, myTimerEvent);
```

```
  // Servo declarations
  esc.attach(8);
  esc.writeMicroseconds(1500);
}

BLYNK_WRITE(V0) {                                   // virtual pin 0 is for setting the midpoint
  int virtualbutton = param.asInt();
  if (virtualbutton == HIGH) {
    val = 512;
    spd = map(val, 0, 1023, 1000, 2000);
    esc.writeMicroseconds(spd);                     // Maps to 512, i.e the zero position
  }
}

BLYNK_READ(V4) {                                    // Refresh rate of 1 Hz
  Blynk.virtualWrite(V4, val);                      // This is necessary because virtual functions cannot be ne5sted for some reason
}

BLYNK_WRITE(V1) {                                   // Increments motor speed value
  int virtualincrement = param.asInt();
  if (virtualincrement == HIGH) {
    val = val + 20;                                 // By 20
  }
}
BLYNK_WRITE(V2) {                                   // Decrements motor speed value
  int virtualdecrement = param.asInt();
  if (virtualdecrement == HIGH) {
    val = val - 20;                                 // By 20
  }
}

BLYNK_WRITE(V6) {
  int virtualslider = param.asInt();                 // Slider for values above 512
  val = virtualslider;
  spd = map(val, 0, 1023, 1000, 2000);               // Triggers new speed
  esc.writeMicroseconds(spd);
}

BLYNK_WRITE(V7) {                                    // Slider for values below 512
  int virtualslider = param.asInt();
  val = virtualslider;
  spd = map(val, 0, 1023, 1000, 2000);               // Triggers new speed
  esc.writeMicroseconds(spd);
}
```

```
BLYNK_WRITE(V3) {                                   // Confirms sending of speed value
  int virtualconfirm = param.asInt();
  if (virtualconfirm == HIGH) {
    spd = map(val, 0, 1023, 1000, 2000);            // Triggers new speed
    esc.writeMicroseconds(spd);
  }
}

void loop()
{
  Blynk.run();                                      // Runs every Blynk functionality
  timer.run();                                      // Runs the timer event
}
```

Actuator Control Code

H_bridgeV2_new

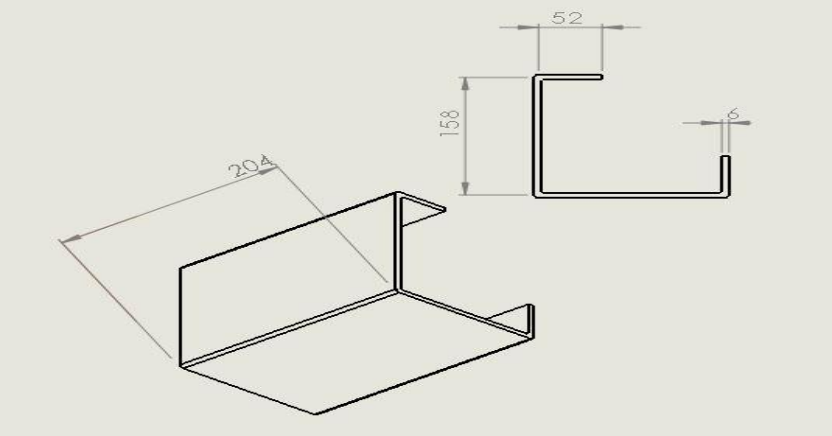
```
// Actuator 1
int dir1PinA = 2;
int dir2PinA = 3;
int speedPinA1 = 6;
int speedPinA2 = 9;                                   // Needs to be a PWM pin to be able to control motor speed

// Actuator 2
int dir1PinB = 4;
int dir2PinB = 5;
int speedPinB1 = 10;
int speedPinB2 = 11;                                  // Needs to be a PWM pin to be able to control motor speed

void setup() {
  Serial.begin(9600);
                                                      // Define L298N Dual H-Bridge Motor Controller Pins
  pinMode(dir1PinA,OUTPUT);
  pinMode(dir2PinA,OUTPUT);
  pinMode(speedPinA1,OUTPUT);
  pinMode(speedPinA2,OUTPUT);
  pinMode(dir1PinB,OUTPUT);
  pinMode(dir2PinB,OUTPUT);
  pinMode(speedPinB1,OUTPUT);
  pinMode(speedPinB2,OUTPUT);
}

void loop() {

  if (Serial.available() > 0) {                        // Initialization of serial communication
    int inByte = Serial.read();
    int speed;

    switch (inByte) {

      case '1':                                        // Motor 1 Forward
        analogWrite(speedPinA1, 255);                  //Sets speed variable via PWM
        analogWrite(speedPinA2, 255);
        digitalWrite(dir1PinA, LOW);                   // FORWARD = LOW on pin 1, HIGH on pin 2
        digitalWrite(dir2PinA, HIGH);
        analogWrite(speedPinB1, 255);
        analogWrite(speedPinB2, 255);
        digitalWrite(dir1PinB, LOW);                   // FORWARD = LOW on pin 1, HIGH on pin 2
        digitalWrite(dir2PinB, HIGH);
        Serial.println("Motor 1 Forward");
        Serial.println("Motor 2 Forward");
        Serial.println("    ");
      break;
```

```
      case '2':                                        // Motor 1 Stop (Freespin)
        analogWrite(speedPinA1, 0);                    //Sets speed variable via PWM
        analogWrite(speedPinA2, 0);
        digitalWrite(dir1PinA, LOW);                   // STOP = LOW on both pins
        digitalWrite(dir2PinA, LOW);
        analogWrite(speedPinB1, 0);
        analogWrite(speedPinB2, 0);
        digitalWrite(dir1PinB, LOW);                   // STOP = LOW on both pins
        digitalWrite(dir2PinB, LOW);
        Serial.println("Motor 1 Stop");
        Serial.println("Motor 2 Stop");
        Serial.println("    ");
      break;

      case '3':                                        // Motor 1 Reverse
        analogWrite(speedPinA1, 255);                  //Sets speed variable via PWM
        analogWrite(speedPinA2, 255);
        digitalWrite(dir1PinA, HIGH);                  // REVERSE = HIGH on pin 1, LOW on pin 2
        digitalWrite(dir2PinA, LOW);
        analogWrite(speedPinB1, 255);
        analogWrite(speedPinB2, 255);
        digitalWrite(dir1PinB, HIGH);                  // REVERSE = HIGH on pin 1, LOW on pin 2
        digitalWrite(dir2PinB, LOW);
        Serial.println("Motor 1 Reverse");
        Serial.println("Motor 2 Reverse");
        Serial.println("    ");
      break;

      default:                                         // turn all the connections off if an unmapped key is pressed:
        for (int thisPin = 2; thisPin < 11; thisPin++) {
          digitalWrite(thisPin, LOW);
        }
      }
    }
}
```

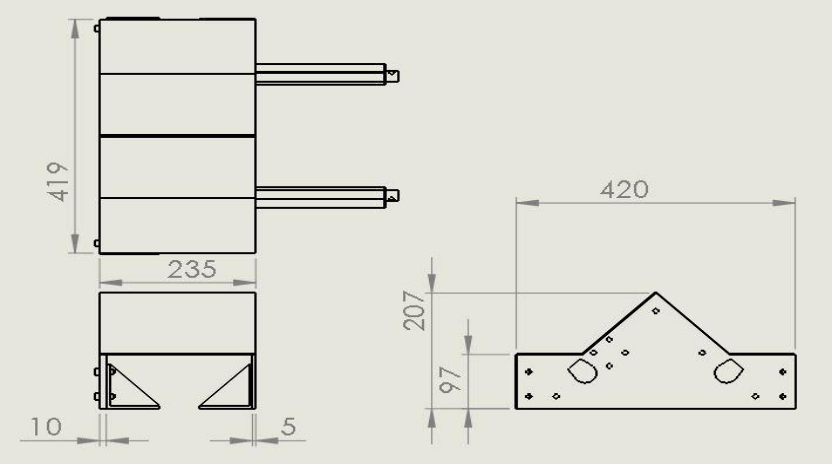
Appendix H – Prototypes and Rough Specifications of Components

Following are the drawings of almost all manufactured components, with their main features: (All dimensions in mm)

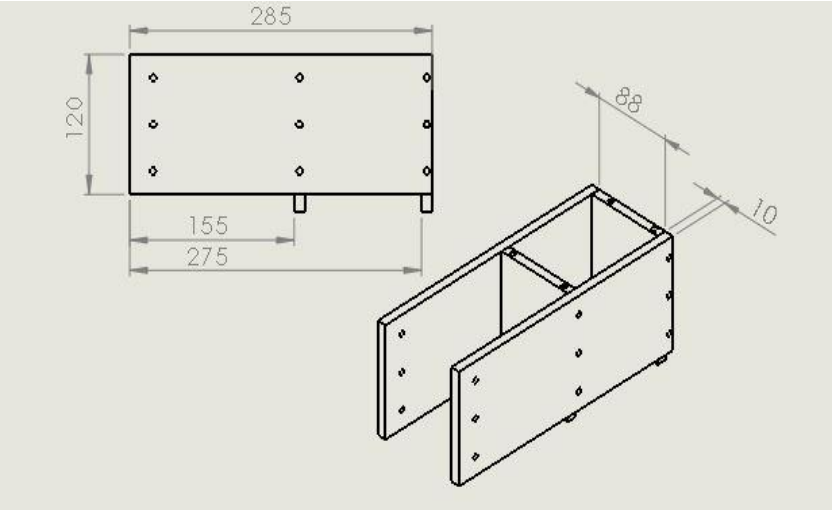
Saddle support unit



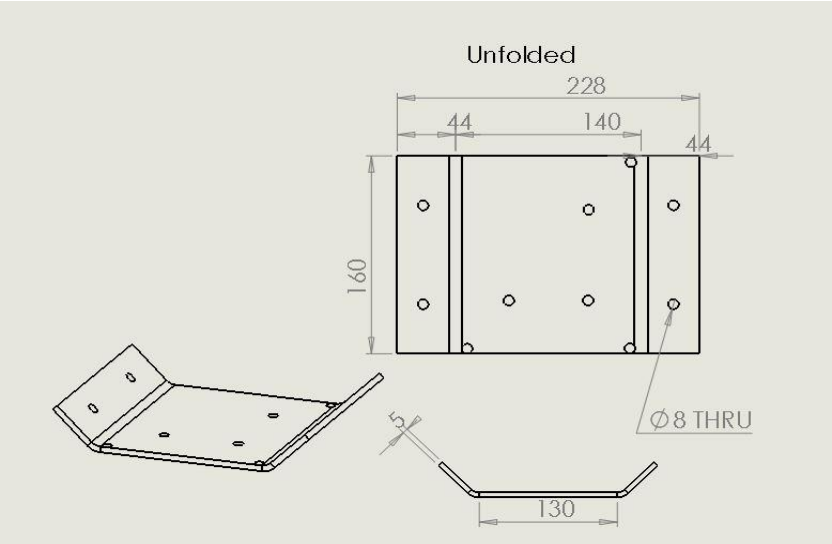
Actuator housing unit



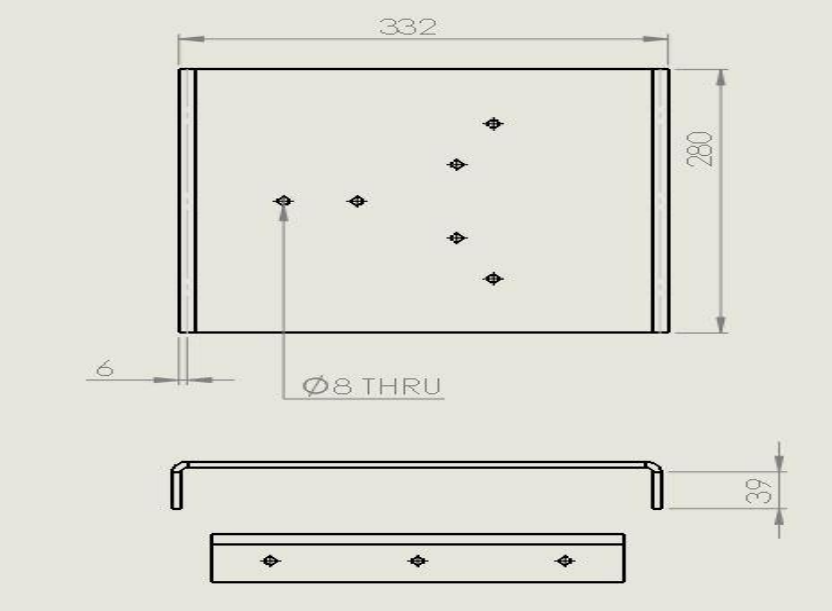
Motor unit



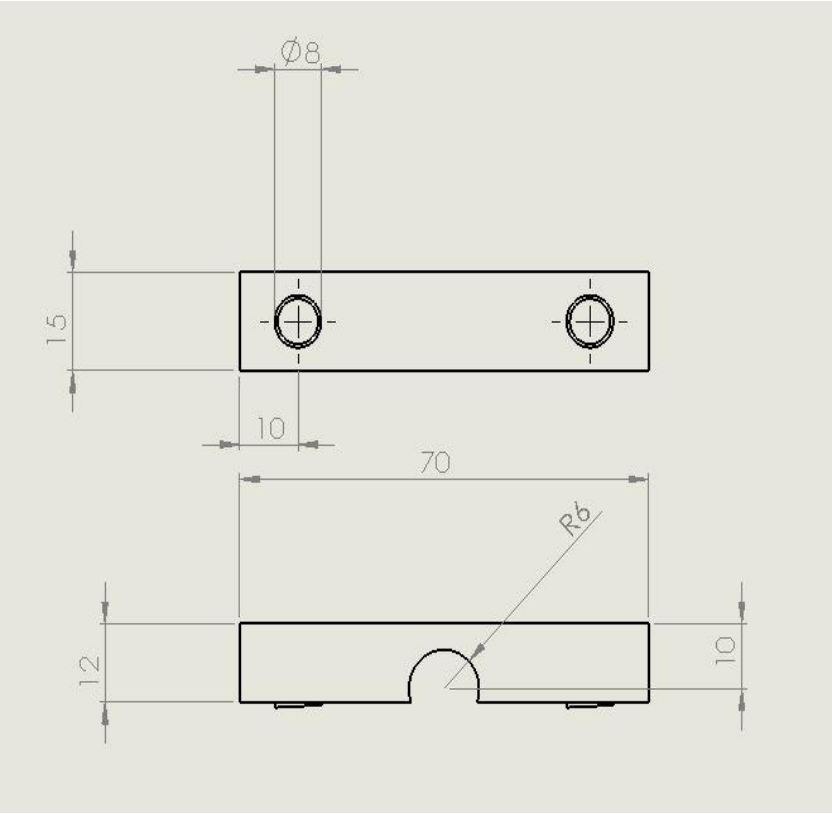
Top spring plate



Bottom spring plate



Spring clamp



Properties of the spring

Free length: 2m
Outer/Mean/Inner diameter: 150/140/130mm
Wire diameter: 10mm
Number of coils: 29.4
Solid height: 294mm
Height in the horse: 800mm
Spring rate:1089 N/m
Material: Chrome silicon steel

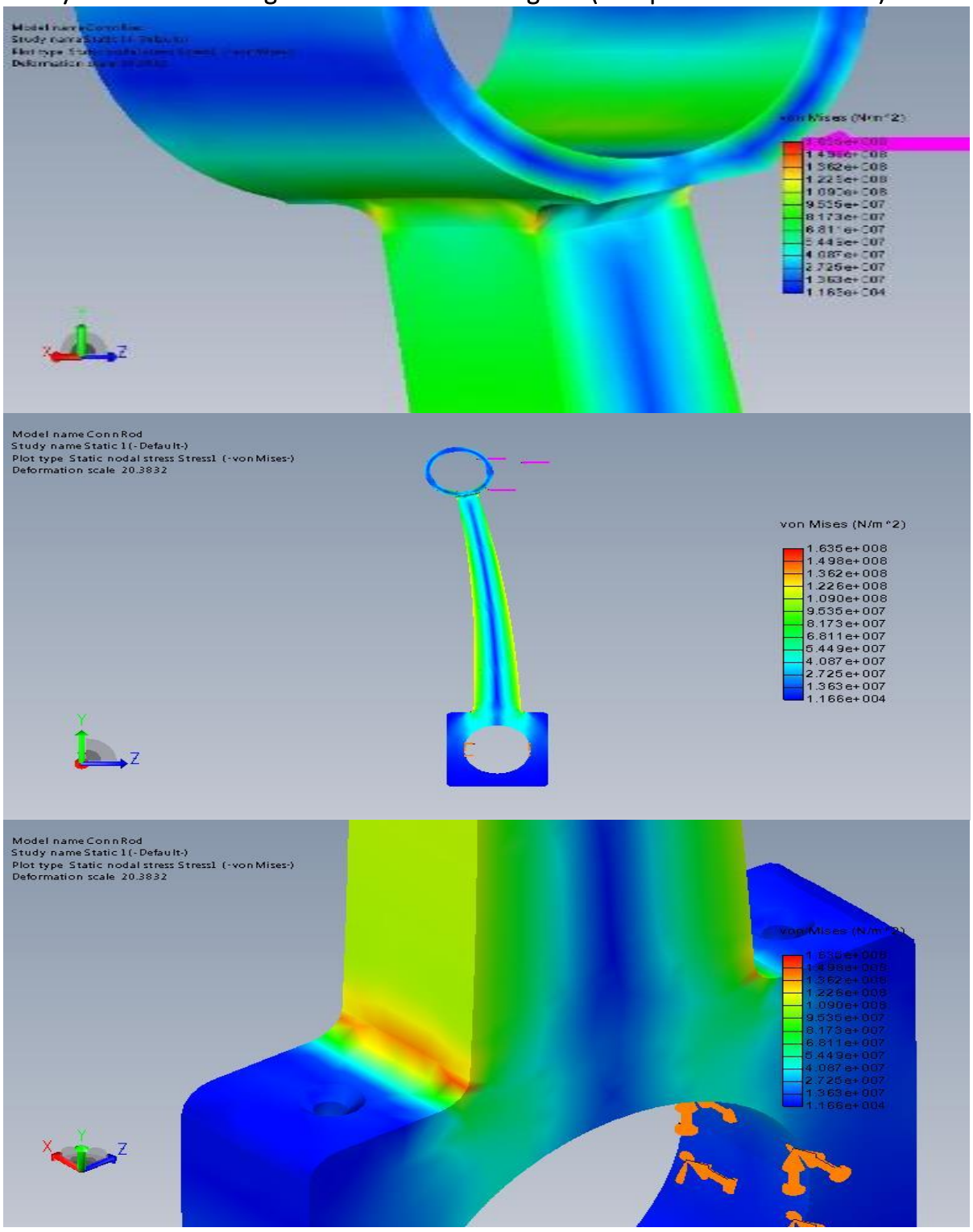
All these characteristics make sure that the spring applies the desired force (1.2 kN) when compressed in the horse, resists to the shear stress and doesn't overly change the applied force as it oscillates (here, while trotting, the force oscillates by +-50N)

Appendix I – Stress Analysis

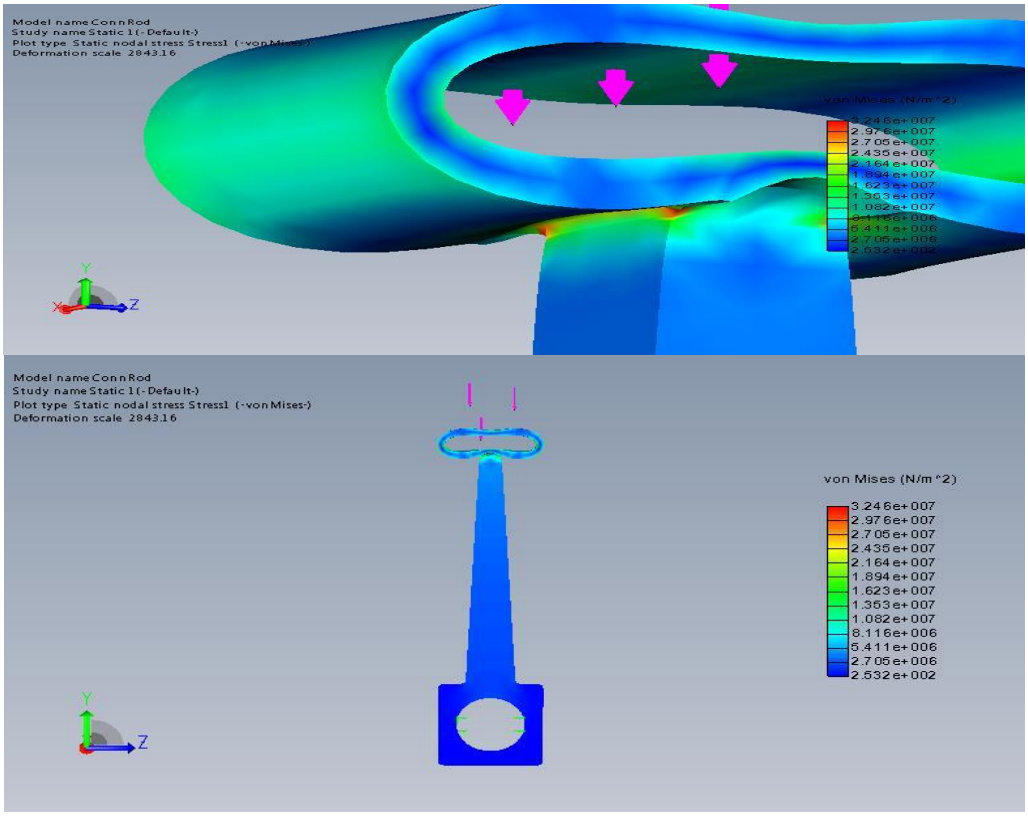
Stress analysis of connecting rods

Because of the new position of the crankshaft, the connecting rods must be ready to take in more load. Because of the complicated shape of the steel connecting rods which might lead to complex stress concentration, stress analysis was run on Solidworks.

Purely horizontal loading of a vertical connecting rod (Low position of the horse)



Purely vertical loading of a vertical connecting rod (High position of the horse)



The yield stress of steel being around 250 MPa (Wikipedia), we can see that even in the maximal acceleration and loading conditions, we have a safety factor of at least 1.5, horizontal loading being the critical position.

Stress analysis of spring

Since the ratio of the diameter of the spring to the wire diameter is big, we neglect any other stress effect than shear stress due to torsion (e.g no transverse shear stress due to axial load or stress due to the curvature of the wire). Thus, the shear stress in the wire is given by[6]

$$\tau = \frac{Tr}{J} = \frac{16T}{\pi d^3} = \frac{8FD}{\pi d^3}$$

where D is the outer diameter, and d the wire thickness.
In our case, with an axial force of 1.2 kN and an outer diameter of 150mm, we have a shear stress of 458 MPa. The minimum tensile strength of chrome silicon steel being 2.07 GPa [7] we have a safety factor of 4.5.

Since the amplitude of the oscillations is very low, we neglect fatigue.
The spring will operate far away of its solid length, where it experiences the most shear stress, so we do not have to worry about that as well, even by taking in account the usual set (creep) of the spring over time which is usually 2% of its length.

[6] Juvinall, Robert C, and Kurt M Marshek. Fundamentals Of Machine Component Design. 1st ed. John Wiley & Sons. Print., Chapter 12

[7] http://optimumspring.com/technical_resources/materials/steel_alloys/chrome_silicon_401_spring_wire.aspx

Appendix J – Ethics

The MechaHorse project enters the realm of a few ethical issues, largely concerning the prevention of excess overlap with prior innovations in horse-riding simulators, and precision and accuracy of our results for safety reasons.

Legality & Respecting Colleagues and Intellectual Property

As our project is both an adaptation and extension of a previous year's project, we were aware of the dangers of outright borrowing elements from the past group. Though we referenced and checked against past data, we gathered our own data to present in our final report and poster, or referenced from external literature.

Furthermore, horse-riding simulators already exist commercially, which we merely used as inspiration. In this case, there was less danger of claiming another's intellectual property as our simulator is targeted towards those with a noticeably smaller budget.

Protection of Human Subjects

Group members undertook a joint risk assessment session with Mr Paschal Egan, in the Bessemer Electronics Lab to ensure each individual understood the risks associated with our project, and various mitigation methods. Each member had also completed induction at the Mechanical Engineering workshop, resulting in working knowledge of the necessary machinery and safety protocol.

We did our utmost to guarantee the safety of our users by ensuring supervision during active simulation, by the theoretical implementation of an emergency system, and by withholding our product from commercial release until it meets the standards and requirements set by the group and the group supervisor.

Project Results: Carefulness & Openness

As the success of our project involves the accuracy of the measured walking and trotting gaits, the relevant calculations and data will be made available to any who wish to verify how closely the simulation matches reality. Similarly, data concerning user safety will be available, to prevent any accidental harm.

Appendix K – Acknowledgements

Many thanks to our supervisor, Dr Warren MacDonald, and our assistant GTA supervisor, Dr Zahra Mohri, for offering insightful feedback whenever needed.

Much gratitude goes to Mr Daniel Nardini and Mr Pascal Egan for their invaluable assistance in the guiding the mechanical and electrical aspects of the final MechaHorse.

Finally, infinite appreciation for those at the Civil Engineering and Mechanical