

MECHTRON 4TB6 - Mechatronics Capstone Projects

**Systems/Component Design**

**Group 8**

Dhruv Aggarwal

Gurkarmjit Kooner

Joseph Moolasseril

Auda Rab

Ahmed Belal

# 

# 

# **Purpose**

The purpose of this document is to provide a more detailed technical overview of the system, from both the hardware and software perspective. For the hardware component of the document, computer-aided design (CAD) drawings will provide context to the overall system. Various figures and diagrams within this document will communicate the breakdown of inputs and outputs of the system. The software component of the document will include pseudo code, module internal specifications and design (MIS, MID), along with diagrams explaining the context of the software system.

# **Scope**

This document will outline the details of the inner workings of the hardware and software systems. Additionally, it also provides an overview of the structure, information hiding, separation and modularity of the system. The component design demonstrates the coverage of individual system parts and reflects the full extent of functionality delivered by the system.

# **Module Guide**

There are multiple modules that make up this small-scale robot which can be broken down into mechanical, software and hardware components. This section will encompass the overall context of the system design. The context diagrams below show the boundaries between the system and the external environment. At the component-level individual subsystems show input and output variables.

# **System Level Context Diagram**

Below is a black-box diagram that shows the monitored inputs and controlled outputs that drive the system.

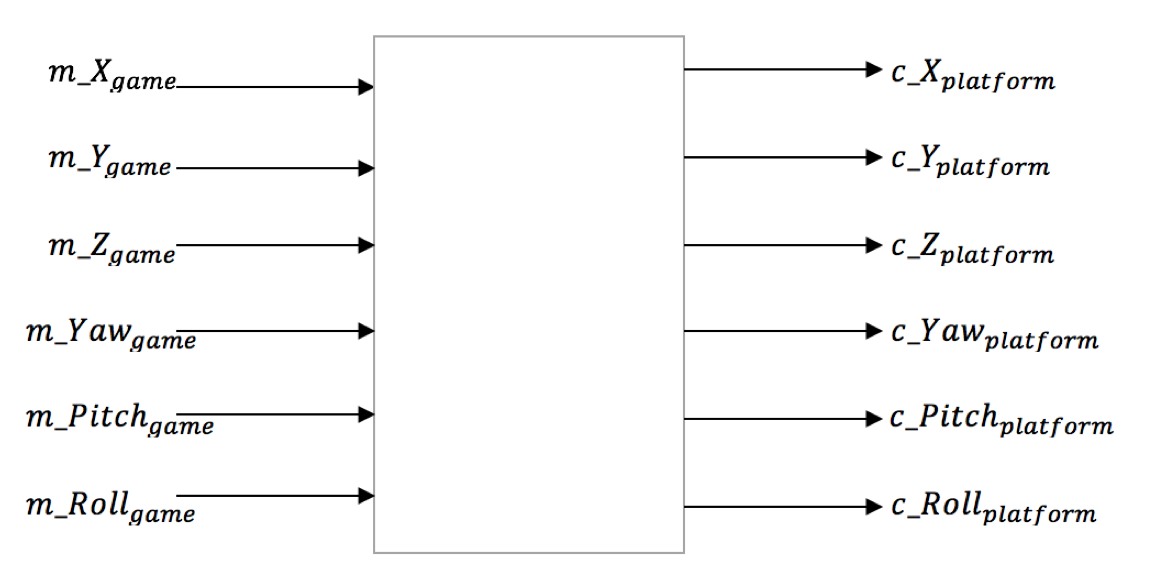


Figure 1 - The Black Box Diagram

|  |  |  |
| --- | --- | --- |
| Variable | Unit | Description |
| m\_X | Newtons | Force on a person in a car in the x-axis (latitudinal direction) |
| m\_Y | Newtons | Force on a person in a car in the y-axis (longitudinal direction) |
| m\_Z | Newtons | Force on a person in a car in the z-axis (vertical direction) |
| m\_Pitch | Degrees | Rotation in the x-axis |
| m\_Roll | Degrees | Rotation in the y-axis |
| m\_Yaw | Degrees | Rotation in the z-axis |

|  |  |  |
| --- | --- | --- |
| Variable | Unit | Description |
| c\_angleServo1 | Degrees | Angle of servo 1 |
| c\_angleServo2 | Degrees | Angle of servo 2 |
| c\_angleServo3 | Degrees | Angle of servo 3 |
| c\_angleServo4 | Degrees | Angle of servo 4 |
| c\_angleServo5 | Degrees | Angle of servo 5 |
| c\_angleServo6 | Degrees | Angle of servo 6 |

|  |  |  |
| --- | --- | --- |
| Variable | Unit | Description |
| k\_X\_Max, k\_X\_Min | cm | Upper and lower limits for translation in the x-axis |
| k\_Y\_Max, k\_Y\_Min | cm | Upper and lower limits for translation in the y-axis |
| k\_Z\_Max, k\_Z\_Min | cm | Upper and lower limits for translation in the z-axis |
| k\_Pitch\_Max, k\_Pitch\_Min | Degrees | Upper and lower limits for rotation about the x-axis |
| k\_Roll\_Max, k\_Roll\_Min | Degrees | Upper and lower limits for rotation about the y-axis |
| k\_Yaw\_Max, k\_Yaw\_Min | Degrees | Upper and lower limits for rotation about the z666-axis |

# **Component Level Context Diagram**

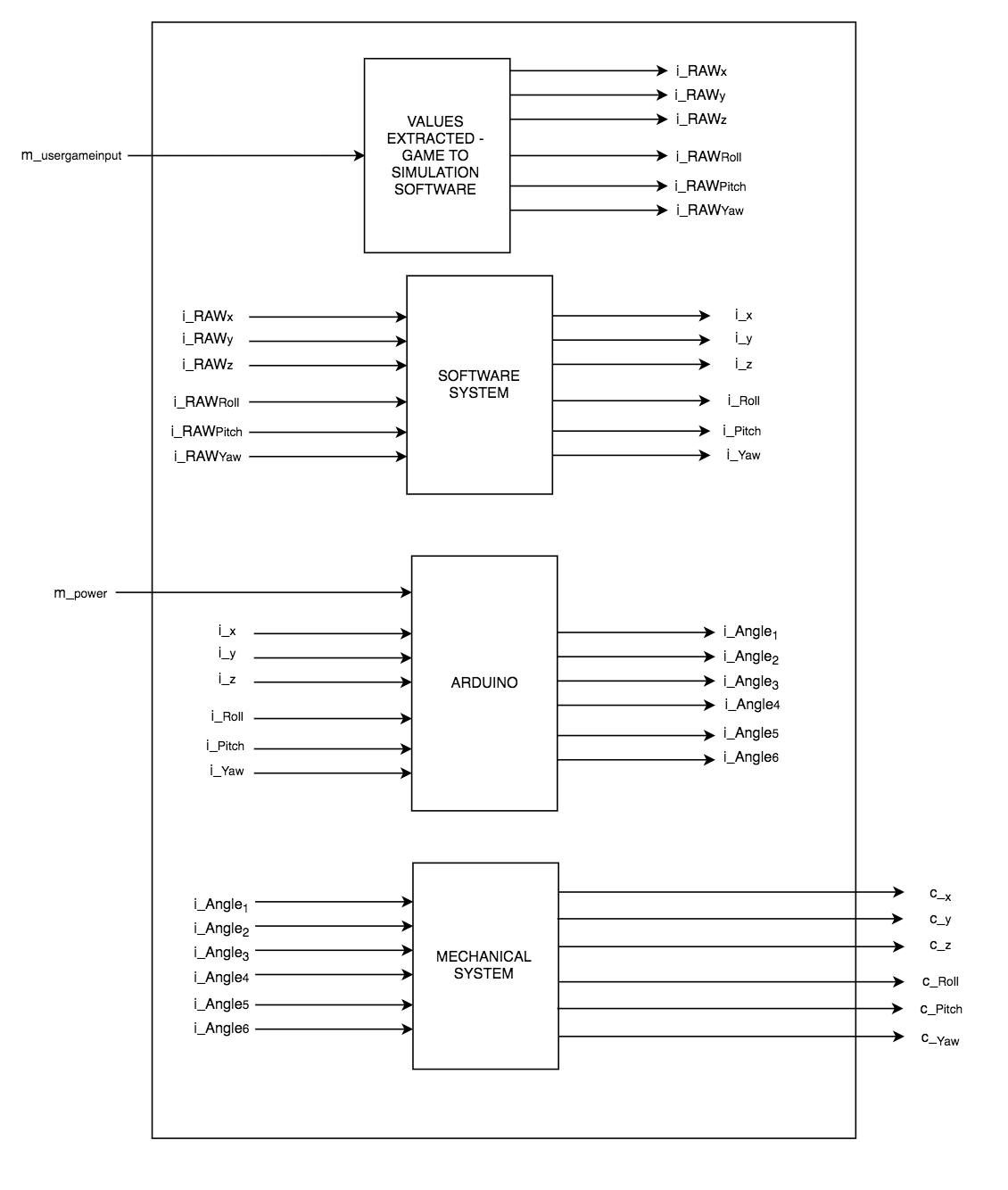


Figure 2 - The Component Level Context Diagram

# **Mechanical Components**

The design of rEX relies heavily on requirements and specifications that were discussed in the Requirements and Goals documents. The chosen design takes into account all the specifications and requirements. However, designing a robot that would meet all the discussed requirements and specifications is not possible due to feasibility constraints. As a result the a small physical model will be designed to represent a full-size physical system.

One of the requirements - “The system shall accurately simulate rotations about x,y, and z” - was a driving factor in determining the design of the mechanical system and the materials that were chosen to build it. Many robot designs were researched and the initial decision was to use a 3-DOF robot which would be able to translate a platform along the z-axis (vertically) and rotate in the x-y plane. After some more research, the NGS group arrived at the conclusion that a 6-DOF Stewart platform robot design would best achieve the project’s requirements by allowing complete freedom of movement (rotational as well as translational) in all directions.

The positions and dimensions of the Stewart platform’s motors, legs, and joints play a vital role in the overall flexibility of movement for its end-effector. The first step in designing the Stewart platform was to determine limits for the platform’s range of movements for its three translational and three rotational degrees of freedom. From there, dimensions for the platform, its base, and its legs were proposed and verified using inverse kinematics to ensure that the design would be able to translate and rotate the platform’s end-effector to the limits of movement desired.[1]

The subsections below will go into detail about some of the manufactured components of rEX. Some building materials not listed here include threaded rods and ball joints which were used to connect the Servo Arms to the Stewart Platform. The threaded rods were cut to exactly 95mm based on the inverse kinematics calculations mentioned above. Ball joints were used to allow three-dimensional freedom in the joints of the Stewart platform to prevent any resistance to the platforms rotation.

**Stewart Platform: Base**

The base of the Stewart platform houses the six servo motors which drive the legs of rEX. The base has specific cutouts to fit the motors into, with holes to screw them securely into place. Mounting brackets are also built into the base to allow for the entire robot to be securely fastened to a larger foundation. Figure 3 and Figure 4 below illustrate the base with and without the servo motors installed. The base is designed so that the axles of the servo motors create a circular shape with a radius of 80mm.

The base was designed using Autodesk Inventor 2018 and then 3D printed. 3D printing the base was the best choice for this part because of its unusual shape and the need for precise angles and placements for the Stewart platform’s motors. 3D printing this part eliminates the possibility for human error with regards to precise lengths, angles, and hole locations.

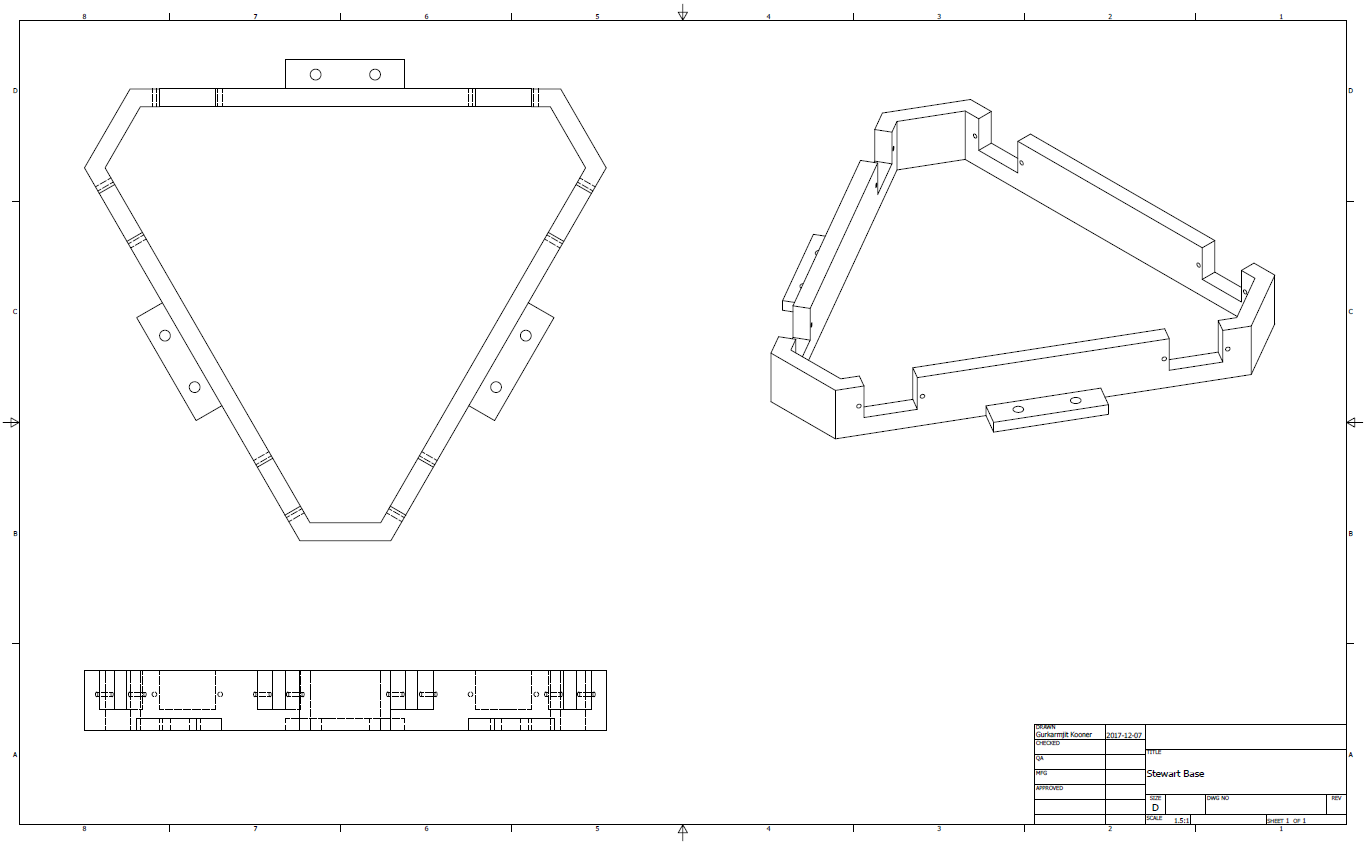


Figure 3 - Base of the Stewart Platform

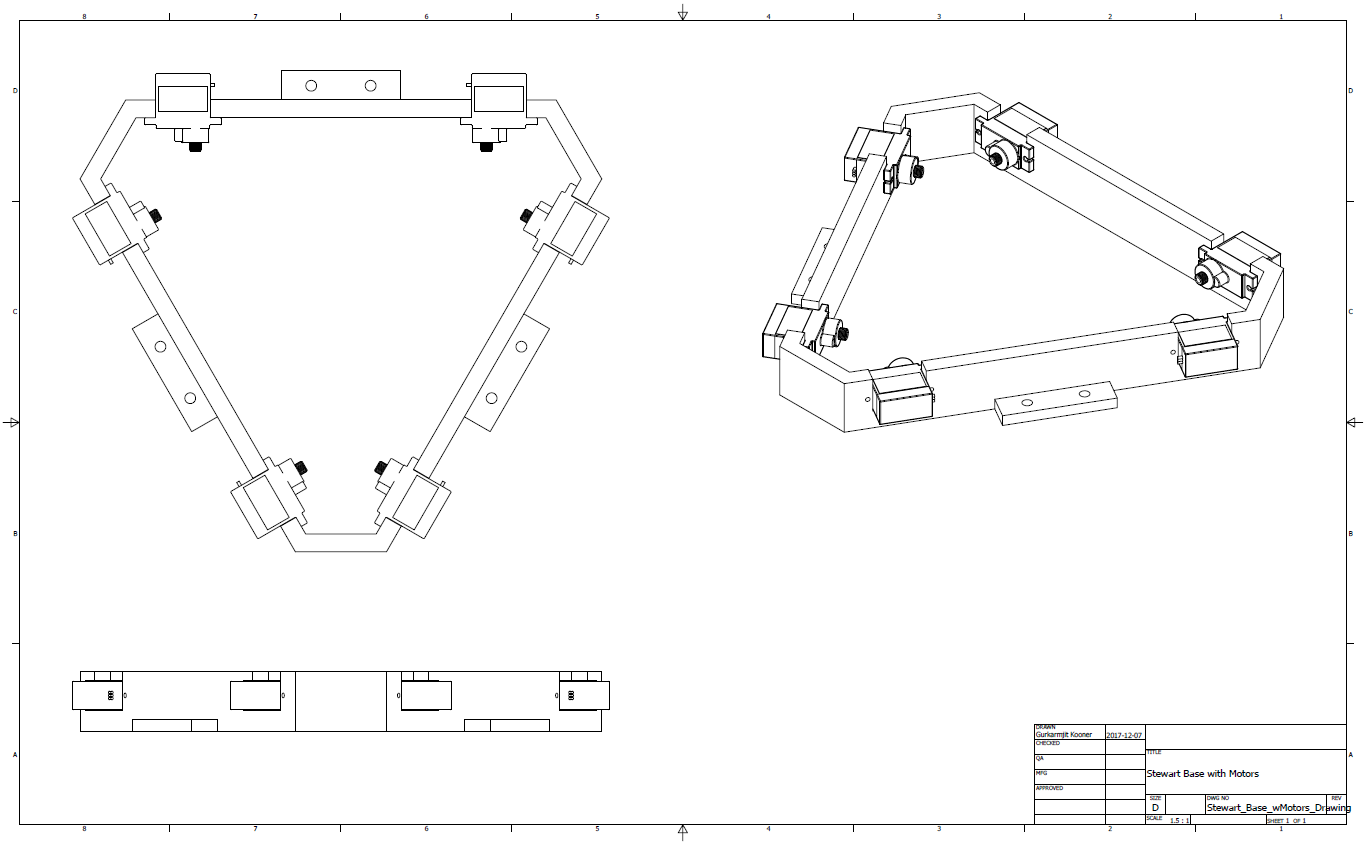


Figure 4 - Base of the Stewart Platform with servo motors installed

**Stewart Platform: Platform**

The platform piece of rEX is the robot’s end-effector. This piece integrates the motions of the six independent motorized legs to create one fluid combination of translational and rotational movements. Figure 5 below shows the bottom side of this platform, where there are holes meant for fastening the threaded rod and ball joints to the servo legs (mentioned in the next subsection). This platform connects the six ball joints into a 50mm radius, similar to the shape of the Stewart platform base mentioned above.

The platform piece was also designed using Autodesk Inventor 2018 and 3D printed. The locations of the holes for this part must also be precise, and 3D printing eliminates the possibility of misalignment from human error. 3D printing this platform was the best option as it created a lightweight, rigid platform with precise measurements and hole locations.

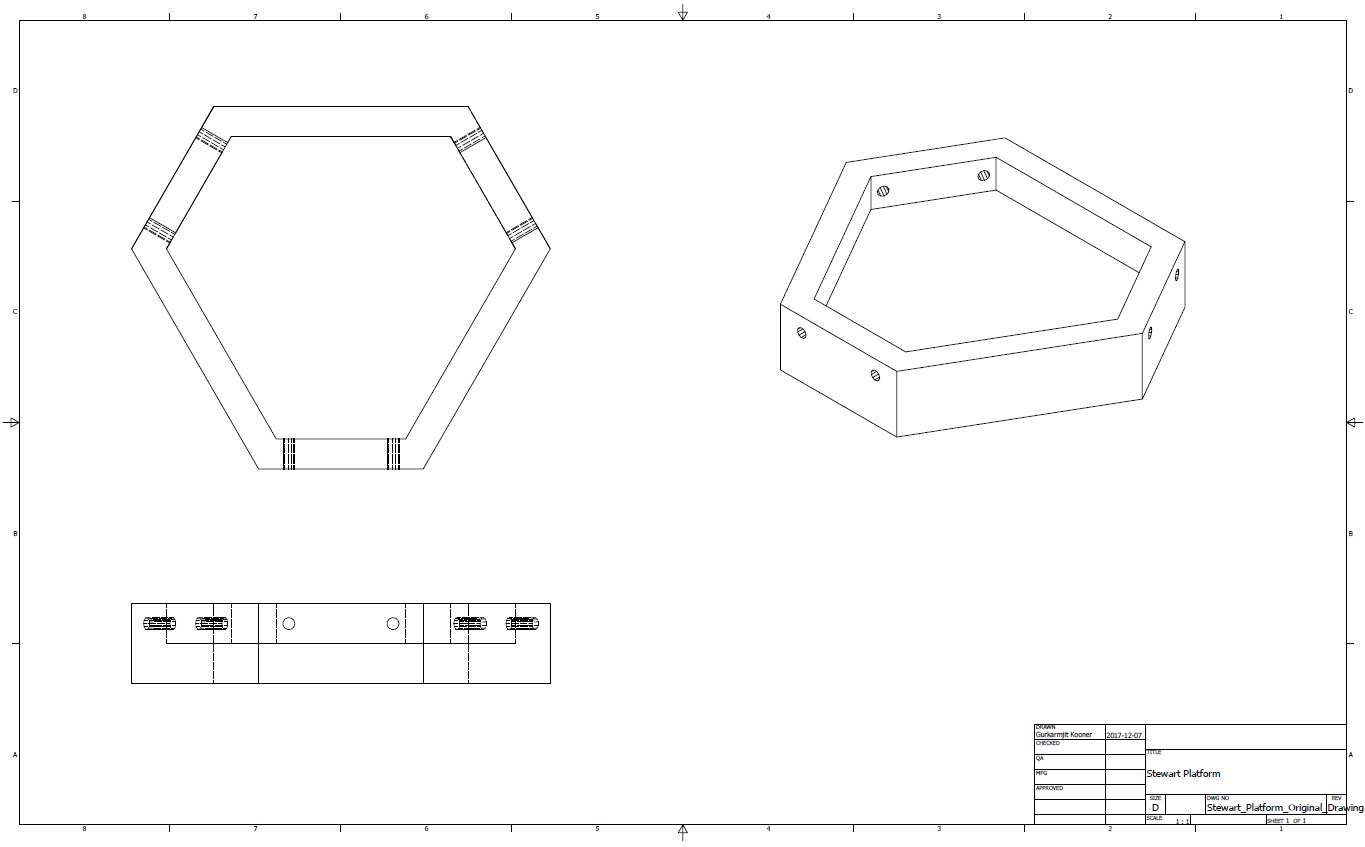


Figure 5 - The platform of the Stewart Platform

**Stewart Platform: Servo Legs**

The servo legs connect each servo to a threaded rod which attaches to the platform. The servo leg piece was specifically designed so that the plastic servo horn that attaches to the motor can be seated within the leg (see Figure 6). This allows the motor to apply larger instantaneous torques which the leg can withstand while remaining securely in place. Furthermore this makes it easier to ensure proper alignment of the zero-degree motor position.

Having the horns sit flush within the servo legs requires precision, which is why the legs were also 3D printed. This also allowed the servo legs to be lightweight, placing less stress on the servo motors. The distance between the servo horn axle and the ball joint measures 45mm as per the inverse kinematics calculations mentioned earlier.

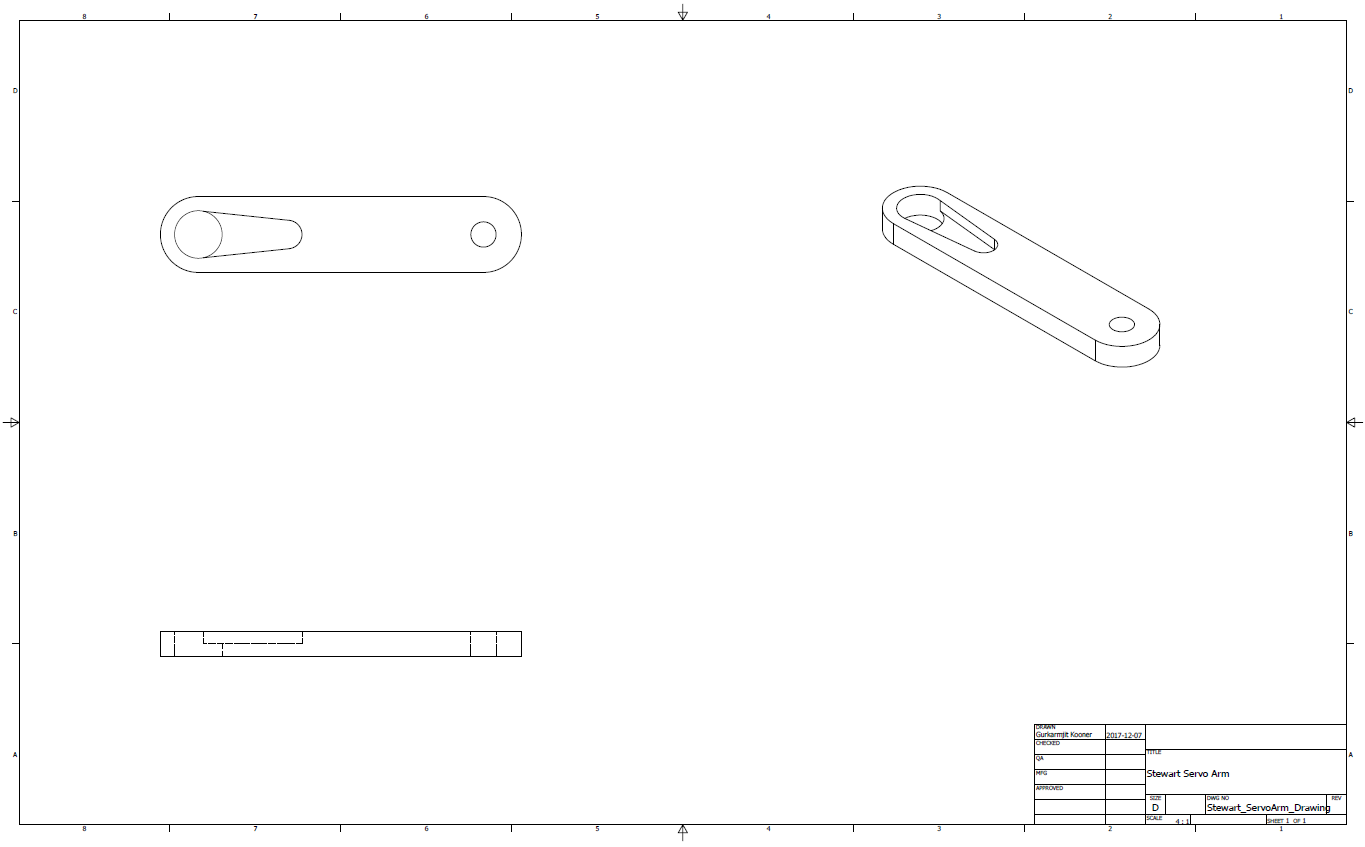


Figure 6 - Servo Legs

# 

# **Hardware Components**

Hardware considerations play a critical role in ensuring that the system can move in a accurate and timely manner as outlined in the system requirements document. The hardware chosen - including motors, a microcontroller, and a computer - are compatible for intercommunication and can operate within the time constraints. The following sections will cover the details regarding the hardware considerations and its importance to the system.

## **Personal Computer**

A significant portion of the software will be compiled and run on a personal computer. This decision is primarily due to the racing video game that will be running while establishing communications with the microcontroller. The computer game will be running on the personal computer which will minimize the intermediary communications between multiple devices allowing for intensive computations and maximizing performance. The following modules are covered by this hardware:

* Robot controller
* Racing video game
* Racing simulator

Inputs:

User’s button inputs from the video game

Output:

Serial data to the robot controller (angles will be sent to the appropriate motors)

## **Microprocessor – Arduino Uno R3**

The Arduino Uno R3 microprocessor is the bridge between the PC and the servo motors. This hardware was chosen because it is able to communicate with the chosen servo motors while providing real-time signals to drive the motors. The video game simulator interfaces easily with the Arduino Uno 3 via serial communication. Furthermore, the Arduino allows for better modularity allowing components to function independently of one another.

Inputs:

Serial data from the PC that provides force values the driver experiences in the video game

* Input signal from the Simulator provides the force values that the driver experiences in the car
* The force value is then converted to degree values using force calculations and algorithms
* Finally, the degree value is communicated to the Arduino using the appropriate formats

Output:

Serial data in degrees is sent to the Arduino to drive the appropriate motor

* Delimited array of current positions in degrees of each is motor is received from the simulator at the baud rate of the USB serial interface.

## 

## **Servo Motors**

Tiny and lightweight with high output power. Servo can rotate approximately 180 degrees (90 in each direction) One motor is placed along each side of the platform for symmetry; this ensures stability and prevents undesirable mechanical deflection/torsion that could otherwise be incurred by driving the carriage from one side only.

Input:

Pulse Width Modulation (PWM) signal from the Arduino controller, for each motor

Output:

Rotational actuation of the appropriate motors based on the PWM signal



# **Software Components**

The software being implemented for the system must execute quickly such that the real-time requirements of the system are met. In order to meet the set out requirements careful consideration must be taken for the software to extract the correct values from the video game and pass them to the hardware components. The software is divided into the following 4 segments, starting from the user input and ending with the motor control.

## **Design Notes**

The primary development of RXSim is in Arduino and responsible for running all the modules through the Personal computer which is an hardware component. Use of X-Sim will also be used for the development however, much of it is going to be for configuration and transmitting to the Arduino.

The significant libraries and tools used to develop the software components are listed:

* X-Sim will provide game data from the video game.
* Arduino will be used as the supervisory system to communicate with X-Sim, Race 07 and motors.
* Race 07 will be the video game that will be used to provide data to RXSim.
* Servo library will be used to provide movements to the servo motors.

## **Video Game - Race 07**

The video game will provide the forces that the user will be experiencing. The video game, Race 07 was chosen for the simulator due to its simplicity and easy to control.

Input:

The user will control the vehicle through keyboard input

Output:

Forces acting on the vehicle, orientation of vehicle, gear, speed and rotation per minute of vehicle

Initialization:

The game will be launched from the simulator which will also start the simulation software

## 

## **Simulation Software - X-Sim**

Input:

The following variables are extracted from the video game.

i\_RAWx, i\_RAWy, i\_RAWz, i\_RAWRoll, i\_RAWPitch, i\_RawYaw

Output:

Filtered data to be sent to the Arduino.

i\_x, i\_y, i\_z, i\_Roll, i\_Pitch, i\_Yaw

The simulation software is crucial in the system as it is the bridge between the racing video game and the Arduino. The selected simulation software provides a direct communication with the Arduino Uno through the USB serial interface. The software is also able to filter and smooth the values returned to ensure a better performance of the system.

Derived Timing Constraints:

Due to the real-time nature of the simulator, data from the software to the Arduino must be passed at a speed that won’t hinder the performance of the simulator.

tserialcom < 50ms

Initialization:

The simulation software will have to be initialized to the profile of the video game. This is to indicate which force and orientation variables are to be extracted while the game is running. The profile also outlines how the axis information will be formatted before it is sent to the Arduino. The expected format is X~a01~ where X represents the axis and a01 represents the data from the axis. The output data is expected to be a decimal value with an 8 bit resolution.

## **Data Conversion**

This section of the Arduino code is responsible for receiving the filtered data from the simulator software serierally and converting the values to angle positions. The code will first retrieve the values from the serial bus and then separate the data based on the axis field defined in the simulation software. The angle positions that are calculated will be used by the motor control section to position the motors.

Input:

The input is the data received from the simulation software.

I\_x, i\_y, i\_z, i\_Roll, i\_Pitch, i\_Yaw

Output:

Angle of servo motors

i\_Angle1, i\_Angle2, i\_Angle3, i\_Angle4, i\_Angle5, i\_Angle6

Derived Timing Constraints:

Data conversion needs to execute at the rate the data is being passed in to avoid delayed responses. The conversions should be completed at the same speed as the serial communication time (50ms). New commands are limited by the serial baud rate.

Initialization:

No initialization is needed for this section. Values are simply passed in and out of the function.

## **Motor Control**

Motor Control will send PWM values through the Arduino pins to the servos. The PWM values are based on the desired servo angles to accurately simulate the vehicle forces. Motor Control receives converted values from the Data Conversion section of the code.

Input:

Angle of servo motor

i\_Angle1, i\_Angle2, i\_Angle3, i\_Angle4, i\_Angle5, i\_Angle6

Output:

PWM value to servos.

c\_x, c\_y, c\_z, c\_Roll, c\_Pitch, c\_Yaw

Derived Timing Constraints:

The timing for Motor Control must be the same as the serial communication (50ms).

Initialization:

Upon start up, the platform must be brought to a home position. Home position will be set as 90 degrees for all servos.

# 

# 

References

[1] “The Mathematics of the Stewart Platform,” Wokingham U3A Math Group, 06-May-2013. [Online]. Available: <https://web.archive.org/web/20130506134518/http://www.wokinghamu3a.org.uk/Maths%20of%20the%20Stewart%20Platform%20v5.pdf>. [Accessed: 22-Oct-2017]

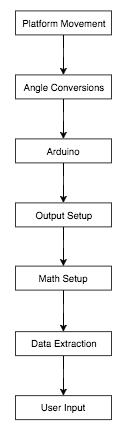
# Software Component Analysis

**Software Module Guide**

Figure 2 outlines the various components in the system. These components can be further broken down into modules, methods, and libraries. The two main computing systems are the PC and the Arduino. The PC is responsible for running the video game and the simulation software. The Arduino is responsible for data conversion and motor control. The following section goes into further detail of these sub-components.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Software**  **Module** | **Description** | **Input** | **Output** | **Component** |
| **X-Sim** | | | | |
| **Data Extraction** | X-Sim will extract the game data based on the users in game decisions | User playing the game | Raw motion Values | Values Extracted - Game to Simulation software |
| **Data Transmission** | X-Sim will transmit data from the video game to the microcontroller | Raw Motion values | A string of refined motion values | Software system |
| **Arduino** | | | | |
| **Data Retrieval** | Serial data to 6 length array of forces | A string of refined motion values | An array of 6 motion values | Arduino |
| **Data Conversion** | Convert numbers to angles (inverse kinematics) | An array of 6 motion values from X-Sim | An array of 6 motion values converted to angles | Arduino |
| **Motor Control** | Setting motor values to the appropriate angles | An array of 6 refined force values | An array of 6 servo angles | Arduino |

**Module Internal Design (MID)**

This section provides further details of modules mentioned in the table above. A high level outline of each modules key attributes will be given. All methods used in each module will be highlighted as well.

Data Extraction/Transmission

*Methods:*

*Math Setup*

* Input - The raw extracted values of X, Y, Z, pitch, roll, and yaw from the video game
* Output - A 32-bit force value that has been passed through spike and smoothness filters

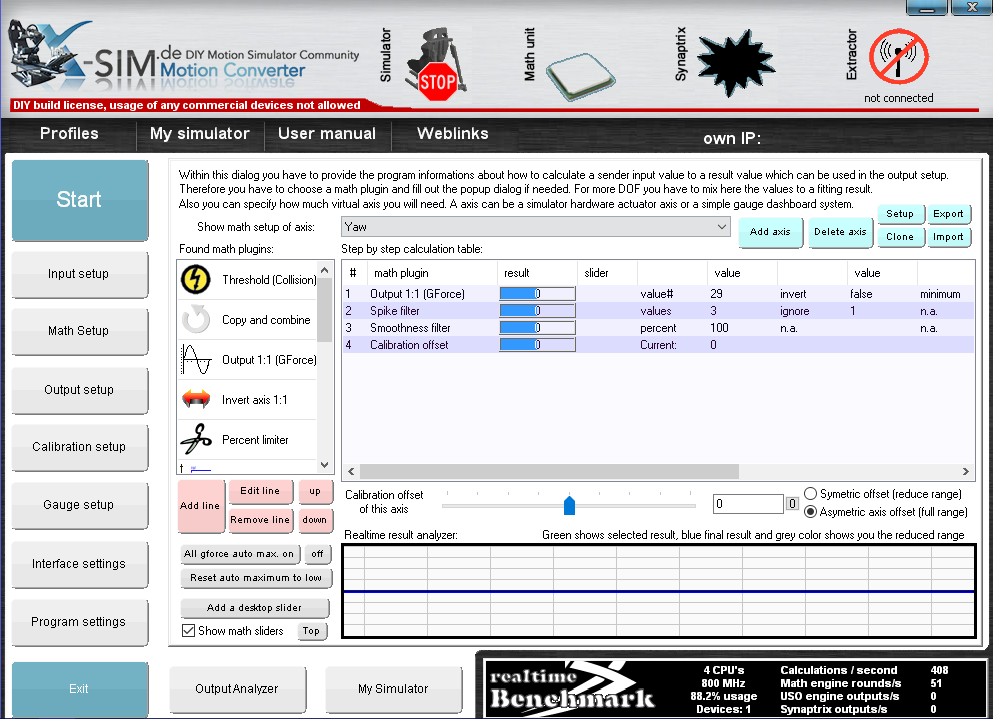


Figure: Math Setup Interface

*Output Setup*

* Input - 32-bit force value from the math output
* Output - The individual forces are combined into a single string that is transmitted to the universal serial output using the Arduino comport at a speed of 115200



Figure: Output Setup Interface

Data Retrieval/Data Conversion

*Methods:*

<float()> serialDataToArray(<string> serialData)

* The data sent via serial communication from X-Sim is stored in Arduino as one concatenated string
* This method splits that string into the original six variables that were transmitted from X-Sim
* Parameters:
  + <string> serialData - string containing the retrieved serial data
* Output:
  + An array containing six elements - the X, Y, Z, pitch, roll, and yaw values (i.e. the orientation variables) transmitted by X-Sim

<float()> arrayToAngles(<float()> orientationArray)

* Performs inverse kinematics equations [1] on a set of orientation variables to determine the positions of the six servo motors
* Parameters:
  + <float()> orientationArray - An array containing the six orientation variables (X, Y, Z, pitch, roll yaw)
* Output:
  + An array containing six angles

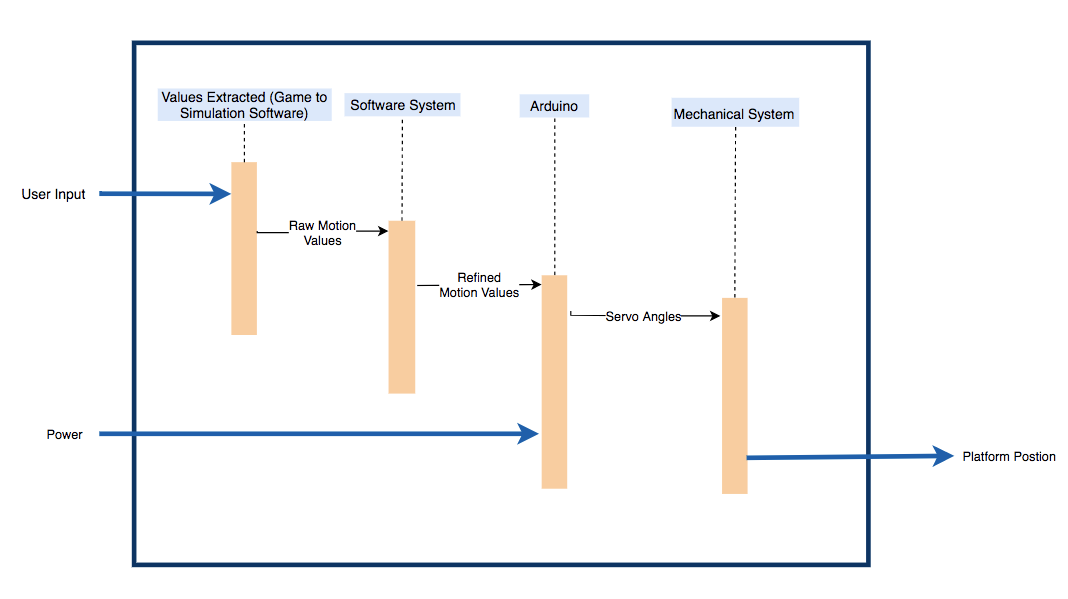
Motor Control

*Methods:*

void anglesToServos(<float()> servoAngles)

* Sets the six servo motors to the servo angles calculated by the previous methods
* Limits the angles to fit within the tolerable range if necessary (0 to 180 degrees)
* Parameters:
  + <float()> servoAngles - An array containing six angles

**Scheduling**



**Conclusion**

This document indicates the current progress on RXSim. The document also breaks down the system into its various components and provides a detailed description of the parts used and justification for the design choices made. The system has been designed to follow the outlined requirements. Future modifications might be made to the system if issues arise or design changes need to be made.