INTRODUCTION

The purpose of this project is to showcase the utilization of the Stewart Platform design for PID control system applications that require quick and precise stabilization in real-time. Some great examples of this technology applied include, docking stations for space shuttles on the International Space Station, flight simulators for pilot training, and military tanks for keeping the barrel completely level while driving over unlevelled terrain. A simpler example for demonstrating this type of system is a ball-balancing robot. A ball-balancing robot is essentially any machine or system that has the ability to stabilize a ball on at least two degrees of freedom.

The position of the ball when stable is usually positioned where all ends or edges of the beam or plane are at equal distance to the ball itself. This is usually the center of the platform, but one can design a system where equilibrium is somewhere else, or the program it in such a way where the system can balance the ball at any point on the platform. These kinds of robots use a variety of different methods to achieve the same purpose. The method primarily depends on how many degrees of freedom the robot will have. The more degrees of freedom, the more advanced the system must be. The simplest version would have two degrees of freedom, a ball moving back and forth on a tilting beam. This is depicted in Figure 1 below:

Where the most advanced version would have a full six degrees of freedom. This version is classified as a Stewart Platform. With applied geometry and kinematics, it is possible to have a platform capable of having full range of motion using a total of six motors or servos. Each servo or motor allows for a degree of freedom. This can be seen in Figure 2 below:

This project is designed to simulate the functionality of a Stewart Platform in reference to the range of motion it can provide compared to other ball-balancing solutions as a proof of concept for my Senior Design Project. Using MathWorks MATLAB and the Arduino development environment, I will be able to show the mathematic proof of a Stewart Platform in action via MATLAB figure reacting in real-time to the mechanical system.
**PID CONTROL**

PID Controllers are an essential tool used in control systems for the purpose of precise control loop feedback systems. This is a technology primarily used in software for applications where there needs to be a set amount of control over multiple variables related to the desired output. Some examples include, managing a temperature range in a delicate environment, executing precise movement from a robotic arm in a factory or surgical room, or cruise control in a car. All of these applications require the ability to control variables that determine how a system reacts to feedback.

PID is an acronym that stands for Proportional Integral Derivative. It is a three-part controller that uses each part to correct the error of difference between the current output and the desired output. In a simple ball-on-beam balancing system, the speed at which the beam will adjust angles, margin of error, and oscillation of the system all depend on which PID constant is being utilized in the control system.

If the system is only using a proportional gain controller, P controller, the only feedback will be the distance of the ball from the desired output, in this case, the center. The ball will constantly oscillate back and forth and will never stop at the center because the speed will increase at each oscillation. This is called an unstable control system.

The D controller takes a derivative of the distance over time, calculating the velocity of the ball. In this scenario, when the ball is moving, the beam will adjust to slow the speed of the ball, stopping it at the position it is in. The new problem is this controller does not care about a desired location, just a desired velocity.

The I part of the PID controller is not a component used by itself. The purpose of this is to adjust the margin of error another controller may have. For example, a PD controller may center the ball and slow it down, but when the ball gets close, the system will be very shaky due to large adjustments for such small distances or speeds. The I controller starts with small increments and increases as time goes on to achieve the desired result. The reason this is not used by itself is because the system will constantly oscillate similar to the P controller.

A completed PID controller is precise, fast, and smooth relative to its respective components. Each system has its own requirements and the constants for each controller component can be determined through trial and error or calculation.

As stated previously, there are control variables for each correction method. These are known as the PID constants. In mathematical terms they are depicted as so; $K_p$, $K_i$, and $K_d$, where all K’s are non-negative gain. In a simple closed-loop system with one $K$, gain is the proportional value between the magnitude of the input and the magnitude of the output. For PID that value is $K_p$. The integral gain is $K_i$ and the derivative gain is $K_d$. There are also other variables that must be considered; error, $e(t)$, and of course time, $t$. These variables can be used to create an overall PID control function, Figure 3, the sum of all gain which is equivalent to the output over the input.

$$u(t) = K_p e(t) + K_i \int_0^t e(t') dt' + K_d \frac{d e(t)}{dt}$$

*Figure 3: Output over input PID controller equation (Source: https://en.wikipedia.org/wiki/PID_controller)*

This can also be represented in block diagram, Figure 4, which is more common in control systems. In the block diagram, $y(t)$ is the output and $r(t)$ is the input. The ratio between the two is the total gain of the system $u(t)$ depicted in the equation above.

*Figure 4: Block diagram of a basic PID controller (Source: https://en.wikipedia.org/wiki/PID_controller)*

The individual gain components are added together at the summation and plus symbols. The value at the output of the first summation
symbol on the left is the error of the system. This value is what is constantly changed throughout the implementation of this controller that is to be corrected at the output.

As previously shown in the ball on beam example, we know that the different PID gain constants of the controller affect the output of the system differently. Usually in a control system, the desire for an output is to achieve equilibrium or a specific value within a relatively reasonable time frame. This is usually depicted by a graph showing the output over a period of time. This output in a control system is commonly referred to as the steady-state response.

On a graph we can plot the steady-state responses with different controllers and gain values. Using MATLAB, we can see the steady-state response, relative to a specific controller at different values to demonstrate what actually happens to something like the ball-on-beam example. Plotting an example steady-state response with two gain variables held constant, while one gain variable is changed and plotted over the other steady-state response plots with the same changing gain variable is an exceptional way to visualize the affect each constant has within a PID controller.

**METHOD**

In order to show the Stewart Platforms range of motion, I recorded basic movement of an Arduino based prototype. In addition, in order to ensure a real-time system, a simulation of the project using one-to-one scale measurements for calculations would be sufficient to demonstrate the range of motion. In order to generate a simulation of how the robot would move in real life, many measurements had to be collected. These were mostly taken from the specification sheets of the selected components. These included the servos, the horn attachments connecting the servo drive axle to the Heim joints, rods, and ball end joints. Any other dimensions had to be assumed in order to have a full simulation because we do not have the part that will hold the motors in place yet.

As seen in Figure 5, there are many points and lengths relating to the platform and base of the robot. Because servos are used instead of linear actuators in this project, the number of variables involving angles double due to the horn of a servo pivoting a rod connected by a Heim joint.

![Figure 5: Dimensions and variables required in a Stewart Platform](image)

Once all these dimensions were collected either by measurement or assumption, we are able to calculate the necessary values using inverse kinematic equations. We can also get these values if a CAD of the robot was created on something like Autodesk Inventor or SolidWorks.

These values can be used to generate a to-scale simulation on a 3D design program. However, since we want to showcase this in action, MATLAB can be used to apply these values to a figure that can then be manipulated with code. Entering our dimensions as variables to apply to inverse kinematics give us the ability to generate a figure on MATLAB that mirrors our physical design, were the red hexagon represents the base, blue shape represents the platform, the joints represent circles, green lines represent distance between the servo output and the spherical end joint of the platform. This is shown as Figure 6 below:
The joints at the base are Heim joints, they only have one degree of freedom, and the joints at the platform are spherical joints which mean they have 2 degrees of freedom. This is figure was simplified in the sense that the green lines represent the hypotenuse of the triangle created by the servo horn and rod, rather than the rods themselves. Although, those are accounted for in the calculation.

The communication between the Arduino and MATLAB is interesting. Arduino usually works by constantly running a code waiting on operation. MATLAB has a code specifically for the control of servos via Arduino. This is done by turning the Arduino into a server awaiting a command from the computer via serial port. These commands come from the code written in MATLAB by the user/programmer. After every action performed by the Arduino, it then waits for another command from MATLAB.

DISCUSSION

The operation of this simulation falls in the two separate codes as mentioned in the Method section. The first code is downloaded from the MATLAB Arduino IO package which allows MATLAB to have control over any servos connected to the Arduino. This code is to remain untouched as it is necessary for MATLAB to properly communicate with Arduino. It is uploaded to the Arduino via Arduino IDE and makes the board a server. Then the code on MATLAB connects to the arduino and identifies what ports it wants to utilize. Then I declared the dimensions mentioned earlier in the code to be later input into trigonometric equations. The parameters that will change in the figure will be the x, y, z rotation and the x, y, z translation of the top platform. These are what move the six moving points shown in Figure 6. The stationary points are on the bottom connected to the base of the robot.

The values are printed the Arduino pins which then output to the servos. Once that data is sent, the figure updates based on the set position of the servo. The figure will not update if the MATLAB notices the Arduino has no servos connected to it.

A running version of both codes are shown below as Figures 7 and 8 (bottom of paper). The Command Window shows MATLAB first attempting to connect to the Arduino server. Once, the server is detected and verifies the code the Arduino is running, MATLAB then runs the rest of the code, which constantly updates the length of the links (green connections between the base and platform). As the servos move, the position of the servo is read by MATLAB, which updates the values used in the inverse kinematic equations, thus updating the link lengths in the Command Window and simulation figure in real-time.

CONCLUSION

Considering the initial plan was to have a working simulation to verify the math related to the function of the Stewart Platform, I would say much improvement is needed. Due to the way I configured the Arduino Uno to MATLAB, MATLAB was expecting real values to be returned so it can update the figure. With no servos attached, it is impossible for MATLAB to change the figure even if the code is sent to the Arduino.

Given more time, I would have changed the code in MATLAB to update the figure based on the calculations performed rather than feedback values of the servos. I am hopeful however, that using 360 degree motion servos that utilize PWM rather than writing a value to the Arduino, I will be able to utilize the majority of this program to properly determine the range of motion our robot will have, given our chosen dimensions. This will
allow us to move forward with determining the proper variables required to design and implement a PID control system.

Figure 7: Running MATLAB code with simulated Figure in Real-Time.
Figure 8: Command Window showing link length and angle updates