Final Report, Group 34

Rigid Body Hexapod Motion Control with Servo Motors for Precise Pointing of Telescope Mirror Segments

Dennis Smalling, Kaustubh Vinchure, Michael Rouleau, Robert Kancans

December 7, 2018

Abstract

This project proposes the use of a modified hexapod as an actuating mechanism for pointing a telescope mirror. Hexapods show great promise in this field as they are smaller and sturdier than traditional actuation systems yet still provide six uncoupled degrees of freedom. Modifications will be made to the traditional 6 piston Hexapod design to improve cost and weight. Prototypes for hexapod actuators have been designed and tested. These designs have been shown to have tilt accuracies on the order of 20 arcsecs. The hexapod will have a pointing accuracy of at least 2 arcmin and a repeatability of 20%

1 Introduction

There has been an increasing push for advanced optical systems within the space industry. Whether for deep space exploration, surveillance, or earth science, customers are expecting optical missions to see farther and clearer than ever before. As such, the precise pointing and control of optical elements has become increasingly important to the space industry. This drive has led to the adoption of numerous mechanisms and techniques previously deemed only reasonable for earthbound technologies. The most significant of these technologies is the Stewart-Gough platform, more commonly known as a hexapod. At its core a hexapod platform is a plate attached to six rigid mechanical linkages. If properly constrained by the linkages, the position and attitude of the platform can be adjusted solely by varying the lengths of the control rods. This type of linkage allows the platform to move in all 6 rotational and translational degrees of freedom while still being mechanically constrained and supported at all times [3]. Most modern hexapod designs achieve this effect by using electronic pistons or other linear actuators as the rigid linkage. By securing one end of the actuator to a fixed base and another to the platform by the use of ball joints, a stable geometric configuration of the limbs can be formed that gives the platform a limited 6 DOF range of motion simply from controlling actuator values.

Hexapods have numerous commercial uses in various commercial and industrial applications. Most notably however is their use as pointing systems in satellite ground station transceivers and observatory telescopes. Hexapods have been widely implemented in these situations due to the fact that they have far superior multiaxis positioning control and noticeably smoother movement than the geared tracking equipment that used to be standard in such systems. Together, pointing accuracy and smoothness of tracking motion represent two of the largest mechanical factors in determining the quality of an image or electromagnetic signal. As such it is these same two qualities that have led spacecraft designers to incorporate hexapods into the newest generation of space based optics and communication equipment.

The issue with adopting ground equipment designs for spacecraft usage is that often ground systems are optimized for very different parameters than are desirable to spacecraft designers. For hexapods this is especially evident in terms of unit mass. Linear actuator based hexapod platforms have become the industry norm due to their durable construction and ability to effectively support the mass of heavy ground based systems. As such most heritage actuator designs have been designed to optimize on the strength of each unit, leading most designs to converge on an incredibly strong (yet bulky) solid metal construction. In the context of a spacecraft, this bulky construction can dominate the mass budget. An extreme example of this is the James Webb Space Telescope. James Webb's design calls for 19 independently controllable mirror segments (18 for the composite primary mirror and 1 for the secondary). Each one of these mirror segments is mounted on its own hexapod, which, by definition, contains 6 linear actuators. This leads to 114 linear actuators being used on the integrated optics package for the telescope, making them the second most common mechan-

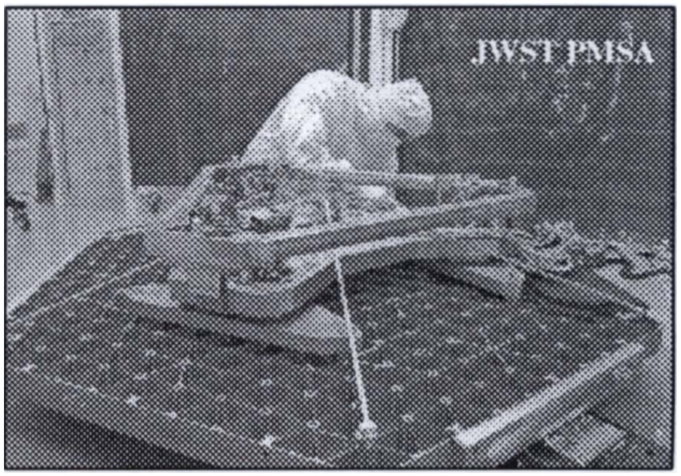


Figure 1: Individual JWST Primary Mirror Hexapod Source: NASA

ical actuator on the entire spacecraft. Since the successful operation of each hexapod is absolutely vital to the success of the overall mission, mission planners decided to use the traditional and proven metal construction on their elements rather than risk experimenting with new materials and techniques [1].

Although a spacecraft the size of James Webb is able to work around the inefficiencies of the current methods, the concern is that smaller spacecraft, such as smallsats and cubesats, operate on a much tighter mass budget and therefore are effectively blocked from achieving the same levels of performance as their larger counterparts. Compounding the issue is the fact that smaller spacecraft are frequently flown by smaller commercial entities and research institutions, meaning that each spacecraft represents a significant investment and therefore most are unwilling to gamble the success of their mission to help certify low TRL hardware. As such a need exists to develop a system that is capable of comparable positioning performance as a traditional hexapod. This system must use existing technologies that have spaceflight heritage, be significantly more mass efficient than a comparable linear actuator based platform, and be scalable to a wide array of platform sizes for different mission profiles.

Servo actuated hexapods have the potential to meet this need. A servo-hexapod consists of the same core design as a traditional hexapod, but instead of having linear actuators to manipulate leg length, legs of a fixed length are connected onto a servo horn. Although the distance between the servo attachment point and the platform remains a constant due to the fixed leg, the distance between the platform and the ground can be adjusted by rotating the servo horn, thus moving the location of the attachment point. This creates a comparable effect on the top platform as if the leg length itself was changed. Although the idea of a hexapod using rotary motors is not new, this type of system has generally been overlooked by most professional applications and is considered a cheap approximation for DIY hobbyists who do not wish to purchase electric pistons or other linear actuators. These types of platforms are never seen in the ground stations and telescope pointers for two primary reasons: firstly is that servos are much weaker at resisting static loading and run the risk of slipping if a heavy weight is applied to the platform. Secondly is that rotary motors add an additional degree of difficulty in determining the position of the top plate. This is due to the fact that the vertical and horizontal placement of the servo-leg attachment point is coupled to the rotational motion of the servo. This greatly complicates the kinematic solution for platform position and therefore there are fears that it will impede the implementation of an accurate control system [2].

In this new field of hexapods for space applications however, these systems may prove more viable than simply cheap approximations. Firstly, servo motors are some of the most common mechanical actuators in modern society. They are cheap to produce, compact, and can be procured off-the-shelf in a wide variety of strengths and form factors. Additionally they have been used in a great number of space based applications, and therefore a wide catalog of parts exist that have proven, reliable spaceflight heritage. Secondly, in a space environment, the limitations on supported platform weight are eliminated due to the microgravity environment.

The goal of this project is to develop a servo-based hexapod pointing demonstrator that is capable of targeting and pointing performance comparable to that required for space based optical systems.

1.1 Design targets

It was determined that for the scope of time and budget of this project, a full reproduction of observatory-quality pointing from the hexapod could not be reasonably achieved. As such a set of intermediate test requirements were devised to establish the performance of the system. In all tests, data collection was managed so that the limiting factors on data resolution were the pixel resolution of the optical elements and the ability of our motors to determine their angular position. Both of these factors can be significantly improved through the acquisition of higher quality components, such as higher resolution cameras and motor encoders respectively.

From consulting with several advisers familiar with space optical systems, it was concluded that a reasonable pointing accuracy with the available hardware would be one order of magnitude coarser than the requirements put on high end pointing systems. As such all test setups were arranged so that the pixel-limited cap on our data resolution would fall within this range. A median value for a reasonable optical telescope grade pointing hexapod was determined to be on the order of arcseconds, from this value an experimental setup was derived where the pixel limited angular resolution value would be on the order of arcminutes. From this experimental setup the following design requirements were extrapolated

1. Pointing Accuracy - 2 Arcminutes
2. Repeatability - 5 Arcminutes
3. Range - 10 Degrees
4. Jitter - 1 Arcminute

2 Design

2.1 Mechanical Design

The hexapod consists of laser-cut acrylic platforms connected by six mechanical linkages, referred to as ”legs”. Each leg consists of an all-thread rod with a ball-joint at each end. Each top ball-joint is screwed into a brass insert that is pressed into the top platform. Each bottom ball-joint is attached to a servo horn which is turned by a servo. Each servo is mounted onto the bottom platform and controlled by an Arduino Uno. The tops and bottoms of each leg are offset from each other to allow the platform to twist as well as tilt. Finally, a laser pointer is press fit into the center of the top platform. The dot from the laser pointer indicates where the platform is pointing.

2.2 Kinematic Model

In order to control the hexapod, the inverse kinematic model must be applied. This series of vector equations translates the six degree of freedom control of the hexapod platform into six servo angles. Knowing the required servo angles allows the hexapod to be pointed as desired. Importantly, these calculations define the range of the hexapod and can be used to predict the constraints of a system before it is physically constructed. The first equation simply relates the length of each leg to the overall position of the hexapod.

$$k_i = T + R_b p_i - b_i \tag{1}$$

Each vector is labeled on the diagram below. R_b simply refers to the rotation matrix of the top platform about the tip,tilt, and twist angles.

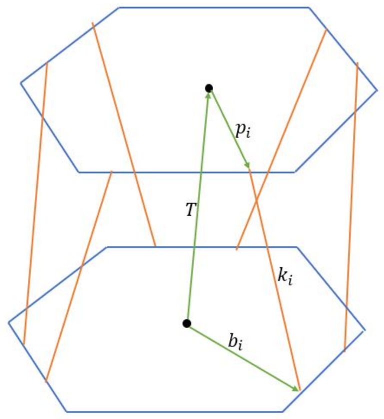


Figure 2: Vector model of the hexapod platform

The next step is to convert leg vectors into servo angles. This is done by using vector algebra to calculate how the servo horn should be rotated to ensure the entire leg is in the correct position.

$$\frac{l_{1i}^2 - k_i^2 - l_{2i}^2}{-2} = k_{ix}l_{2ix} + k_{iy}l_{2iy} + k_{iz}l_{2iz} \tag{2}$$

This equation relates each vector calculated in equation (1) to the actual leg segment that is controlled by each servo. Figure (3) shows how this equation is applied to an individual leg. Each leg is modeled as a triangle which changes it's shape in three dimensions as the leg is moved.

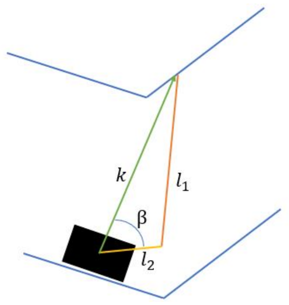


Figure 3: Vector model of an individual platform leg

Each component of each l_2 vector is a unique function of servo angle which depends on the leg's location on the hexapod. This equation is solved graphically to yield the servo angles.

At this point it is important to address the physical reality of the hexapod. In order to ensure that the kinematic model was accurate, the real-life dimension of the hexapod were plugged back into the model to ensure it's accuracy. This included the xyz coordinates of each attachment point for each leg, as well as the real rotation limits for each servo (each servo did not rotate the nominal 180 degrees). Thus the mathematical model was tailored to match the reality of the system. This ensured that the hexapod would be accurate enough for the control software to correct for.

2.3 Control System

Since the goal of the system is to point a laser pointer at a target, the feedback for the system is error from target point. The feedback is in terms of pixel distance from the target pixel. This generalizes the control system over various distances. By determining Δ_x and Δ_y of the measured pixel value of the laser pointer from the target pixel value, two PID controllers are run in parallel to determine the new commanded position to be sent to the hexapod. The PID controller update frequency is limited by the rate at which the servos repeat. The control system will update 3 times a second. Each update will have to allow enough time for the servos to reach their commanded position and for the code to run.

- The camera frame $\hat{x}$ and $\hat{y}$ have to be aligned with the hexapod $\hat{x}$ and $\hat{y}$
- The camera is held still over the test.

The first assumption is necessary to ensure that the commanded motion through the code aligns with the required motion of the hexapod. The second assumption is necessary to use pixel values to correspond to a real world target the laser pointer ought to point to. The implemented control system uses only proportional control. Targets are selected through a GUI that shows a single frame from the webcam.

2.3.1 Detecting the Laser Pointer

The scope of our project was to create a pointer. Therefore, we had to track where the laser pointer was. In order to do this, we had to manipulate the image in order to rapidly detect the location of the laser pointer. Since the detection had to be be both accurate and fast, we used an open source Python library, OpenCV. OpenCV provided image manipulation functions - such as blurring and sharpening an image - that we used to develop a method of tracking the location of a laser pointer. We developed two forms of tracking systems for the laser which used either colour-based or intensity-based tracking, used to identify a target and the laser pointer respectively. The following is the method used to post process images and detect a laser point using an intensity based tracking. Here, the image is first cropped around the last known point - this was neccesary to improve the

Algorithm 1 Intensity Based Tracking of a Laser Pointer

```
1: procedure DETECTDOT(frame, lastPosition)
2:     croppedFrame ← cropAroundLastPosition(frame,lastPosition)
3:     blurredFrame ← blur(croppedFrame)
4:     sharpenedFrame ← sharpen(blurredFrame)
5:     maskedFrame ← threshold(245,255,sharpenedFrame)
6:     imageContours ← getCircularContours(maskedFrame)
7:     laserPointerContour ← mostCircularContour(imageContours)
8:     laserPointerCentroid ← getCentroidFromMoments(laserPointerContour)
9:     return laserPointerCentroid
```

efficiency of the detection. This also minimized noise from appearing in the detection of the dot. For example, a momentary glare could trick the detection into tracking the glare. Afterwards, the image was blurred. This technique managed to remove majority of glare that appeared on the screen and took care of background noise. By blurring, small disturbances in the image could be removed. In addition, this had the benefit of homogenizing the colours and reducing the variation in pixel intensity. Afterwards, the image was sharpened to construct a denoised version of the previous image. Since the laser pointer would always appear as bright white, since the laser pointer was far too intense for the webcam to even resolve its red colour, the conclusion was reached that the pixel would always be the brightest object on the screen. Thus, by masking the image by checking for pixels with intensities in the range 245-255, we can see the center of the laser pointer as well as some of the edges. With the larger contour, determining the center was found to be more reliable. This method found to provide a detection accuracy within a pixel. Below a picture of the object detection detecting the laser pointer center,

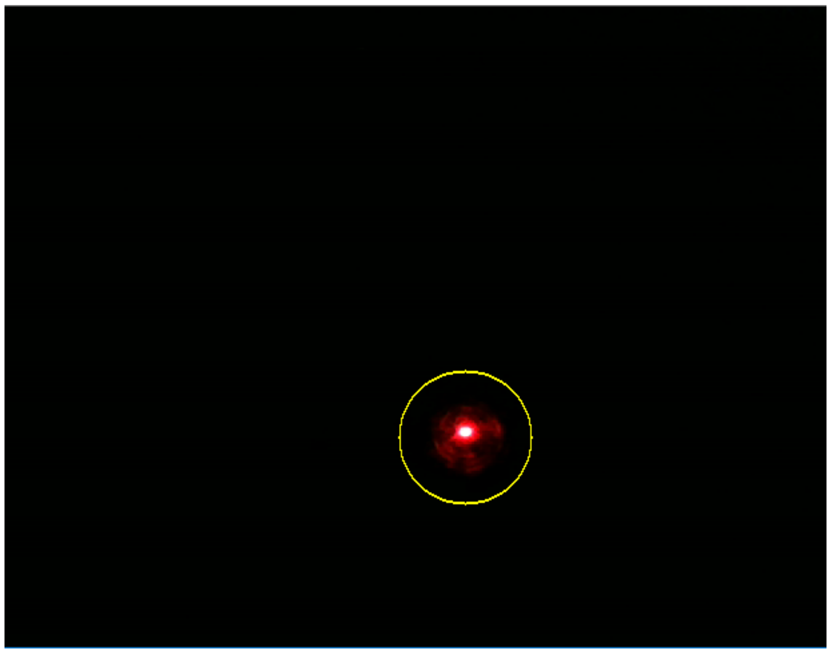


Figure 4: Laser Pointer detected using Intensity-Based Tracking

2.3.2 Determining Distance

Our initial design for the control system was for it to work for arbitrary distances. However, when converting a pixel distance to travel, an estimate of distance is necessary. Rather than measure the real distance, a distance estimate in pixels was used. The distance is determined by initially rotating the hexapod 2 degrees about $\hat{x}$ and measuring the number of pixels the laser pointer moved by. Using the relation,

$$d = \frac{\Delta y}{\tan \theta} \tag{3}$$

where Δy is distance laser pointer travels in pixels, θ is the angle rotated about $\hat{x}$ and d is the distance the surface is away in pixels. While it is possible to use the PID Controller without a robust estimate of distance, it reduces the reliability of the system since the motion is much less predictable. Determining the angle of rotation becomes less accurate since the conversion from distance pixel ought to move and hexapod ought to rotate is dependent on the distance the hexapod is away from the target

2.4 Controlling the Hexapod

Since the Computer Vision was written in Python and controlled through the computer, it was necessary to open serial communication with the Arduino Uno as opposed to using the Arduino Uno as an embedded system. In order to send a command, the handling code was ported onto the Arduino to unpack a package of servo positions and assign it to the correct servo. A motor controller in Python was able to reliably send a packet of servo positions to the Arduino to reliably control the hexapod. The delay in sending and receiving the package was dominated by the delay in moving the servo. Thus, the one third second delay between commands allowed for the servos to move to the commanded position.

3 Experimental Setup

3.1 Range

Since the range of motion of the platform is independent of the laser pointer, the test set up did not require the camera to record pixel position. An inclinometer was placed on the top of the platform aligned with $\hat{x}$ and $\hat{y}$. The hexapod was then commanded to rotate about $\hat{x}$ and $\hat{y}$ until an error was thrown on the software side that indicated one of the servo motors were being commanded to go further than 180°. Below is a schematic showing the test setup.

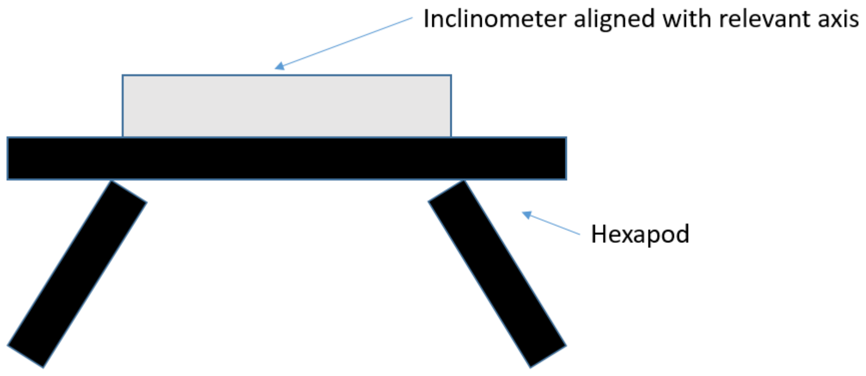


Figure 5: Schematic of Hexapod Range Test

3.2 Repeatability

The camera is placed near the hexapod looking at the laser pointer. The hexapod is commanded to rotate about $\hat{x}$ and $\hat{y}$ at the same angle. By recording the position of the laser pointer as it travels back and forth between the two points, both the path the laser pointer takes as well as the place where the laser pointer points at given the platform position is recorded. The test setup below is used for all tests where the position of the laser pointer is tracked.

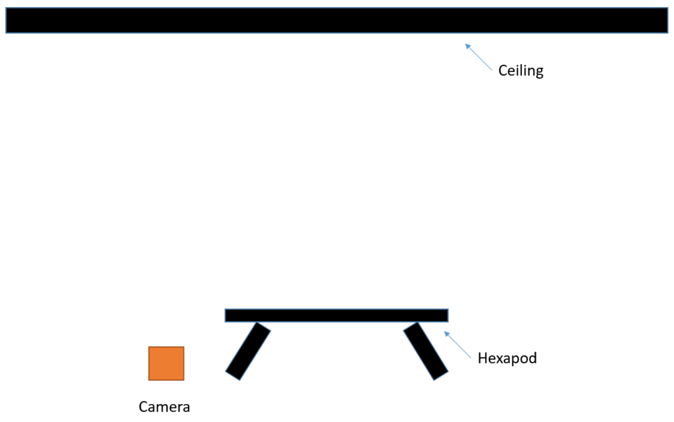


Figure 6: Test setup used for all subsequent tests

3.3 Accuracy

The accuracy test measured the reliability of the feedback system to detect the position of the laser pointer. This detection accuracy had to be within a pixel to ensure that the hexapod was not limited by the software, but rather the resolution of the camera that is used. In order to test this, the laser pointer was aimed a wall and held steady. The software was then commanded to detect and plot the position of the laser pointer on the wall. Below is a schematic of the test,

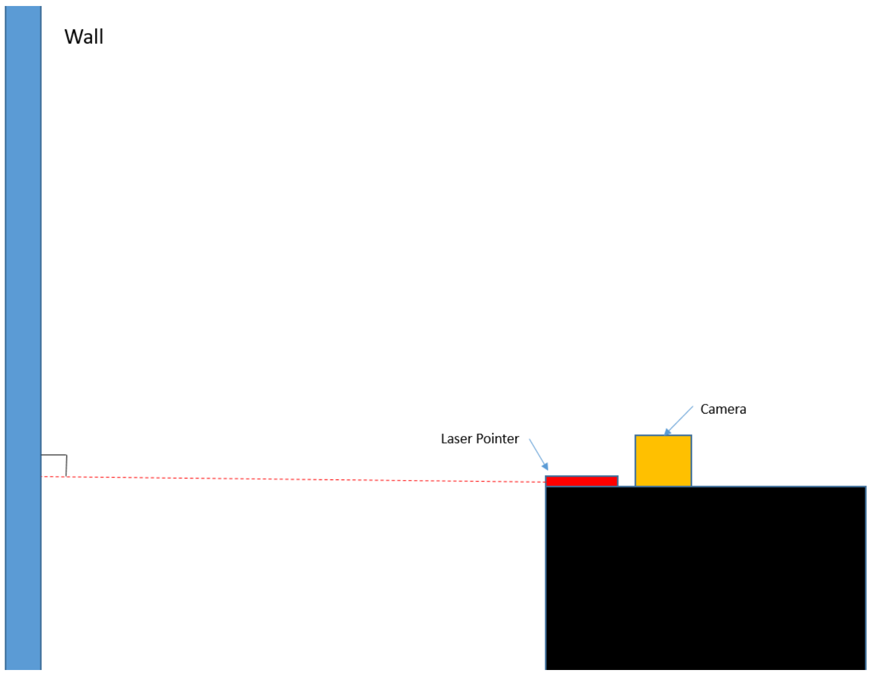


Figure 7: Test Setup used to measure object detection precision

3.4 Predefined Motion Test

One of the concerns we had about the design of the hexapod was that the path the pointer takes between two points cannot be controlled. Solving for the servo angles and moving the pointer between two points produces a path between the two that tends to be repeatable. This is a conclusion we drew from the results of the repeatability test. In order trace out a path, a circle was defined by a discrete set of points. Given the distance between the hexapod platform and the ceiling, the respective position vectors for the hexapod were calculated. Using the same test setup in Fig. 6, the position of the laser pointer was plotted.

3.5 Control System - Hold

In order to put the whole system together, and test the ability of the code to continuously detect the laser pointer, computer the new position, command the hexapod then determine the new position of the laser pointer, the hexapod was commanded to stay held onto a point as the hexapod was disturbed. While this test is qualitative in that it the disturbance of the platform is not a predefined motion, the variance in laser pointer position from the target position can me measured. This test was conducted on a ceiling that was approximately 30 feet away. A secondary goal of the test to was to prove ability to scale up the object detection and pointing capability since earlier tests were conducted on a ceiling 6 feet

3.6 Control System - Point

In addition to being able to hold onto a point, the control system had to show the ability to point the laser pointer to a target, in addition to holding on the target. In order to do this, a GUI was developed that allowed a user to select a location in a picture and convert that to a pixel location. In addition, the control system was designed such that after the laser pointer was within 1 pixel of the target for 5 seconds, it was finished. This allowed us to capture any instance of the laser pointer moving beyond the target pixel. The 5 second condition gave atleast 15 updates for the control system. The test setup in Fig. 6 was also used for this test - with a higher ceiling.

3.7 Jitter

The data during the 5 second hold on the target was used to determine jitter. With the control system still running, the data collected would show the response to the uncertainty in determining the laser pointer position due to the control vision code. The test setup in Fig. 6 was used for this test since it was collected from the Control System - Hold test.

4 Results

4.1 Range

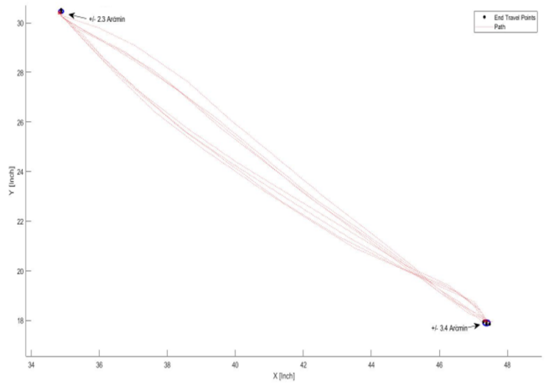
The ranges of motion are given in the table below,

	$\hat{x}$	$\hat{y}$
Maximum	$5.9° \pm 0.1$	$6.0° \pm 0.1$
Minimum	$-5.3° \pm 0.1$	$-6.7° \pm 0.1$

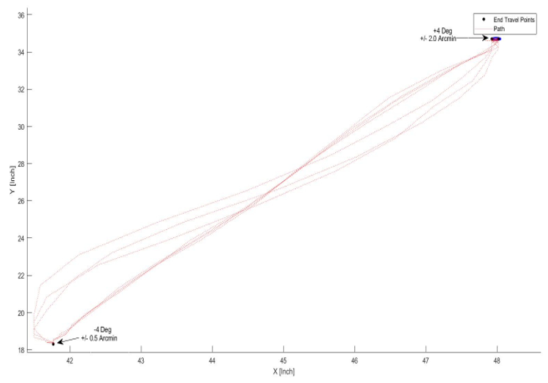
These values are the maximum and minimum values the hexapod would rotate to consistently. This was also used to verify the true maximum and minumum angles the servos would rotate to by measuring the angle of the servos using a protractor.

4.2 Repeatability

The servo rotated about $\hat{x}$ and $\hat{y}$ from -4°to 4°. This tested repeatability of the system as it reached the edges of its performance. In addition we tested motion from 0°to 3°to test the systems ability to return to its home position (where the platform is parallel to the base) and move to a target point. The paths are presented below,

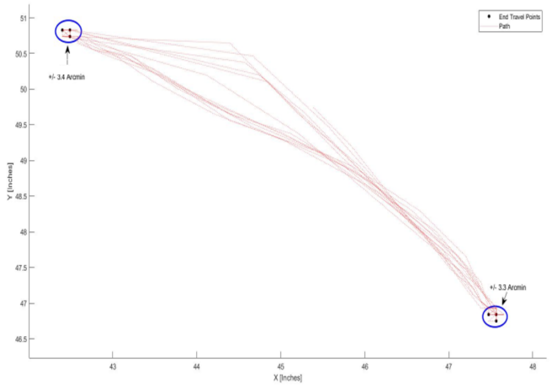


(a) Motion about $\hat{x}$

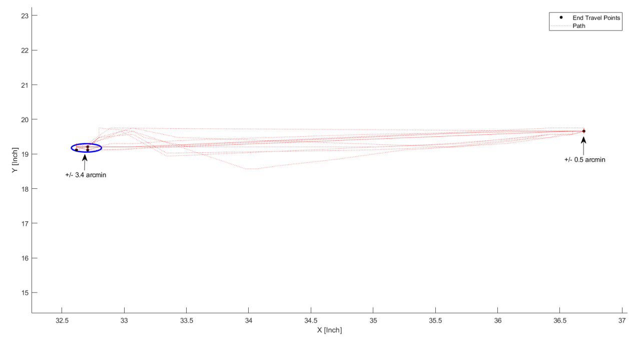


(b) Motion about $\hat{y}$

Figure 8: Hexapod path when rotated to edges of range



(a) Motion about $\hat{x}$



(b) Motion about $\hat{y}$

Figure 9: Hexapod path when rotated to edges of range

The uncertainty in reaching the commanded position are reported in arcminutes below for the test along the edge of the range,

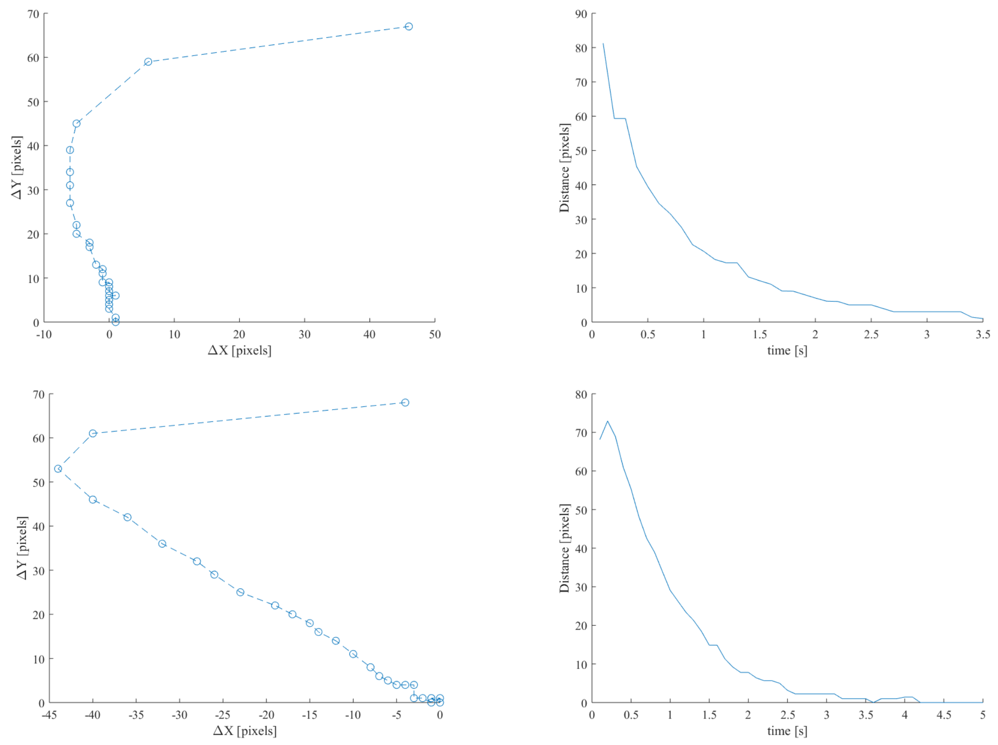
$-4°$ about $\hat{x}$	$4°$ about $\hat{x}$	$-4°$ about $\hat{y}$	$4°$ about $\hat{y}$
±2.3 Arcmin	±3.4 Arcmin	±0.5 Arcmin	±2.0 Arcmin

For the motion to a point far away and to the home position,

$0°$ about $\hat{x}$	$3°$ about $\hat{x}$	$0°$ about $\hat{y}$	$3°$ about $\hat{y}$
±3.4 Arcmin	±3.3 Arcmin	±3.4 Arcmin	±0.5 Arcmin

4.3 Accuracy

The initial distance to the target point from the starting laser pointer position was kept within the range of motion of the hexapod. The distance to the target point in pixels and the position of the laser pointer in the frame were plotted to show both the path and response of the control system.



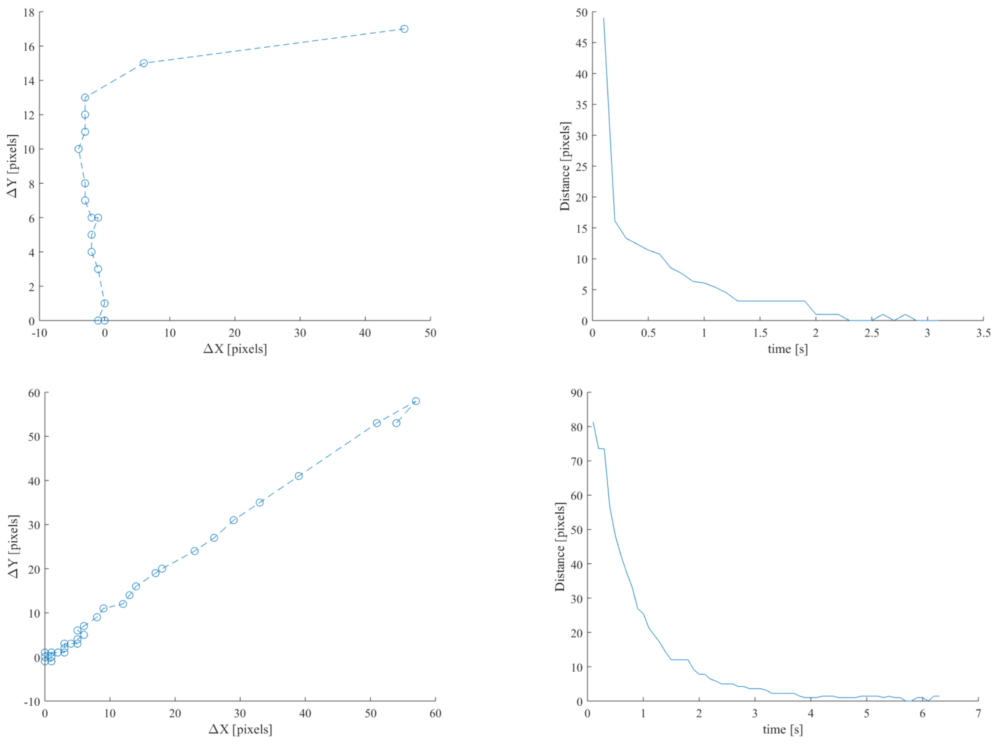


Figure 11: Control System Response and corresponding path taken for each run

For this test, the distance between a pixel corresponded to a 2 Arcmin (0.0006 rad). Below are the convergence times for each run,

Run	Convergence Time [s]
1	3.5 ± 0.1
2	3.2 ± 0.1
3	3.2 ± 0.1
4	3.1 ± 0.1
5	2.0 ± 0.1
6	3.9 ± 0.1
7	3.5 ± 0.1

4.4 Jitter

Aiming the hexapod at a point using the control system also yielded useful data regarding jitter. After the hexapod had converged to the target point, the control system was kept running. By measuring the laser position during this time, we measured the jitter. Below are plots of jitter during the time when the hexapod had settled onto a point.

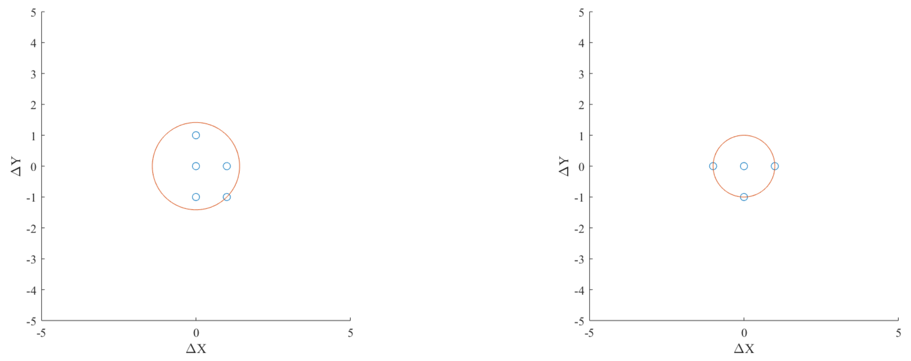


Figure 12: Pointing jitter about the target point

In the plots, the tracked positions are within a pixel of the target position. This one pixel distance corresponds to a jitter of 2 arc-minutes. This is ultimately limited by the resolution of the camera itself, since the camera cannot resolve changes in position under 1 pixel, or 2 Arcmins. When the platform was disturbed, it was visually observed that the laser pointer was able to stay within a pixel of the target. When the platform was tilted and moved the laser pointer further away from the target, the control system was able to settle the laser pointer back on the target. This was visually observed.

4.5 Path Tracing

In order to see the ability of the hexapod to follow a path, a set of 75 discrete points were created, along with their corresponding platform positions. Below is a plot of the path taken with a moving mean average over half a second for clarity

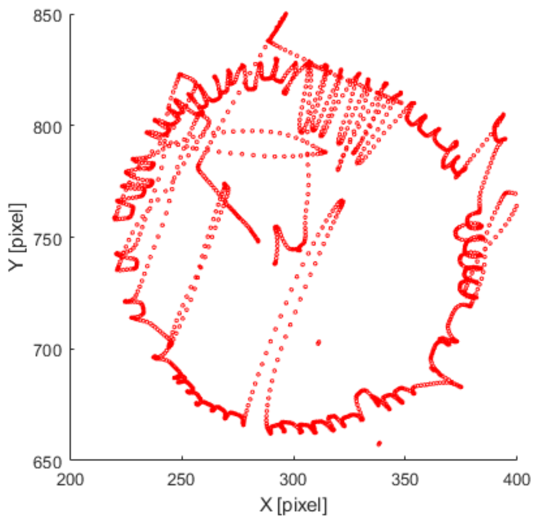


Figure 13: Moving mean average of laser pointer circular motion

The path it takes from point to point is not uniform - created points where the hexapod moves in an unexpected direction before settling on the new point. This is evident in some of the paths taken in the center of the laser pointer. In addition to plotting the moving mean average, the places where the laser pointer settled on are shown below along with the expected path the laser pointer was to take,

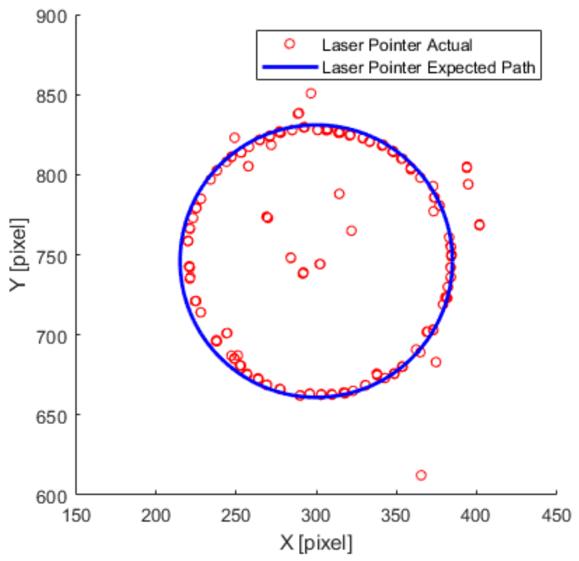


Figure 14: Expected path plotted over points laser pointer settled on

5 Discussion

5.1 Range

Our data indicates that our design target for the range of the platform is within the range of the platform. We designed it to have $\pm 10^\circ$ about both $\hat{x}$ and $\hat{y}$. We measured the range to be $11.2^\circ \pm 0.2$ about $\hat{x}$ and $12.7^\circ \pm 0.2$ about $\hat{y}$. The range of motion of the hexapod is limited by the length of the servo arms in the design. In order to increase range of motion larger servo arms would be necessary. However, increasing servo arm length would also reduce the accuracy of the pointer since the step sizes of the servo motors would still stay the same at 1°

5.2 Repeatability

The platform was proven to be repeatable within the resolution of the camera. This demonstrates that the mechanical design was sturdy enough to eliminate any noticeable slop from the system. Additionally, it verifies that the kinematic model does in fact give exactly one platform angle solution for each set of servo angles. This also gives a baseline for the best possible resolution for pointing accuracy. The measured repeatability for each of the runs is within the 5 Arcmin requirement.

5.3 Accuracy

The accuracy of the hexapod was also shown to be within the limits of the measurement hardware. This successfully demonstrates that a closed-loop control system can be used to point the hexapod at a given point. This software was able to compensate for small imperfections in the mechanical system and ensure that the hexapod was able to be used for very fine pointing. It is important to note that although in this case computer vision software was used to generate feedback for the control system, depending on the situation, another feedback sensor (such as an interferometer, profilometer, etc.) could be used. This could potentially increase the measure-able accuracy of the system. The time the system took to point at a target was reasonable. In actual application, the slew rate of the sensor is not as important, but rather the accuracy is more relevant.

5.4 Jitter

The pointer was shown to be stable when having reached the target point. The distribution of points was contained with 1 pixel of the target pixel. However, a single pixel is also the precision of the computer vision system to determine the position of the laser pointer. In addition, getting a precision less than a pixel is impossible considering resolving differences less than a pixel will not be visible on the webcam. This can be solved by using a higher resolution camera to provide a smaller pixel size. In addition, the control system's performance to hold on a target was validated by ensuring it stayed on target even when the platform was distrubed. Our design target was to achieve 1 Arcmin of jitter. However, we were limited by the resolution of the camera which led to our measured jitter to be 2 Arcmin.

5.5 Path Tracing

The data shows that predicting the motion between two discrete points is impossible. We see that there is a general trend of pointing between the two points shown below,

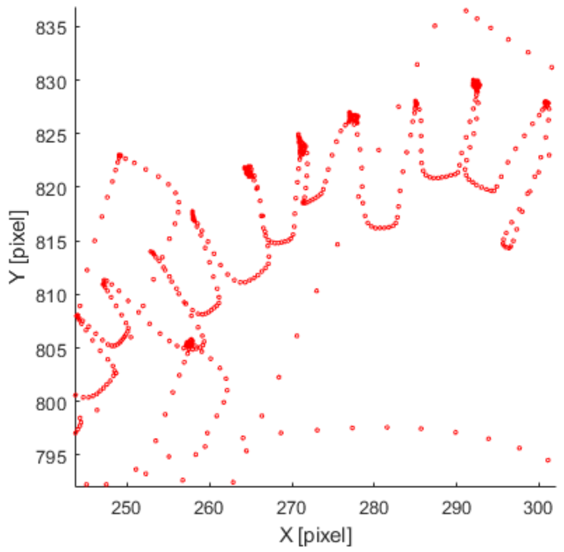


Figure 15: Path taken from point to point for section of circular path

We can see here that while the path taken from point to point is similar, the extent to which the pointer moves towards the center varies over various sections. This indicates that by further discretizing a path, these variations can be reduced and a smoother path can be achieved. Furthermore, we prove that discretizing a path is a valid way to trace a path in Fig. 14. The expected path aligns closely with the points where the laser pointer settled at. There are some time where the point to point path moves much further towards the center, and then moves out. In addition, the starting points from where this test was started are also seen in Fig. 14. The outliers a result of unpredictable motion from point to point.

5.6 Future Direction

The first and most glaring next step is to procure higher fidelity equipment, particularly a better camera. As the system is currently limited by the hardware, notably the camera resolution, increasing optical quality will allow for further characterization and iteration upon design of the platform and control system. Furthermore, this control system may be further iterated on specifically to create an integrated system, but a more compact method of measuring platform orientation, such as high-resolution interferometers, will be required.

The system constructed is scaleable in a variety of factors in order to match most design conditions. The top platform shape may be changed to match any optical element or mirror segment so long as the legs still come to an equilateral triangle to fully constrain the platform motion. This means the system may adapt for primary optics, such as Webb's mirror segments, secondary optics, including a circular parabolic mirror, or for load-bearing cases in space. The platform was observed to be able to withstand a static load of roughly 2.6 kg, but was never run under a loaded condition, therefore a necessary future step is to quantify its response under load.

6 Conclusion

The most important conclusion to draw from this experiment is that the hexapod was ultimately limited by the measurement hardware. It was the resolution of the camera that limited how accurate the hexapod was measured to be. This means it is extremely likely that more expensive, higher resolution cameras would allow the hexapod to be even more accurate in it's pointing. Similarly, servo motors with a precision more precise than one degree would upgrade the physical limitations of the hexapod. Essentially, this hexapod design can be increased in accuracy and precision simply by purchasing more expensive hardware. This is especially true when it is coupled with the closed loop control system. This scalability has very interesting applications for industry. It means that this design is extremely useful in a wide variety of applications. Whether a cheap, coarse pointing system is sufficient, or a more expensive, fine pointing system is required, this hexapod design has been shown to be highly efficient and effective.

References

[1] Allison Barto et al. "Actuator Usage and Fault Tolerance of the James Webb Space Telescope Optical Telescope Element Mirror Actuators". In: SPIE Symposium on Astronomical Telescopes and InstrumentationSpace Telescopes and Instrumentation. Amsterdam, Netherlands, 2012.

[2] Qiang Meng et al. "Dynamic modeling of a 6-degree-of-freedom Stewart platform driven by a permanent magnet synchronous motor". In: *Journal of Zhejiang University SCIENCE C* 11.10 (Oct. 2010), pp. 751–761.

[3] D. Stewart. "A Platform with Six Degrees of Freedom". In: *Proceedings of the Institution of Mechanical Engineers* 180.1 (1965), pp. 371–386. DOI: 10.1243/PIME_PROC_1965_180_029_02. URL: https://doi.org/10.1243/PIME_PROC_1965_180_029_02.

7 Appendix

7.1 Computer Vision

Dot object has a method to detect its position via intensity based and colour based tracking. This is provided below,

```python

from imutils.video import VideoStream
import numpy as np
import cv2
import imutils
import time
import math
from collections import deque

class Dot:
    COLOR_RANGE_TOLERANCE = 50
    def __init__(self,x,y,r,lower_bound,upper_bound):
        self.x = x
        self.y = y
        self.r = r
        self.lower_bound = np.array(lower_bound)
        self.upper_bound = np.array(upper_bound)
        self.cntR = 0
    def getX(self):
        return self.x
    def getY(self):
        return self.y
    def getR(self):
        return self.r
    def setColor(self,lower_bound,upper_bound):
        self.upper_bound = np.array(upper_bound)
        self.lower_bound = np.array(lower_bound)
    def detectDotColor(self, frame):
        hsv_frame = cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
        sharpen_kernel = np.array([[-1,-1,-1],[-1,9,-1],[-1,-1,-1]])
        sharper_frame = cv2.filter2D(hsv_frame,-1,sharpen_kernel)
        mask = cv2.inRange(sharper_frame, self.lower_bound,self.upper_bound)
        #mask = cv2.GaussianBlur(mask,(5,5),0)
        im,cnts,heir = cv2.findContours(mask.copy(),cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
        cv2.imshow('frame',mask)
        if (len(cnts) > 0):
            c = max(cnts,key = cv2.contourArea)
            ((x_circle,y_circle),self.cntR) = cv2.minEnclosingCircle(c)

            M = cv2.moments(c)
            if (M['m00'] > 0):
                self.x = int(M['m10']/M['m00'])
                self.y = int(M['m01']/M['m00'])


        area_measured = self.cntR**2 * math.pi
        return area_measured
    def detectDotIntensity(self,frame):
        gray_frame = cv2.cvtColor(np.uint8(frame),cv2.COLOR_BGR2GRAY)
        gray_frame = gray_frame.astype('uint8')
        blurred_frame = cv2.GaussianBlur(sharper_frame,(5,5),0)
        sharpen_kernel = np.array([[-1,-1,-1],[-1,9,-1],[-1,-1,-1]])
        sharper_frame = cv2.filter2D(blurred_frame,-1,sharpen_kernel)
        ret, threshold = cv2.threshold(sharper_frame,245,255,cv2.THRESH_BINARY)
        cv2.imshow('Thresholded Image',threshold)
        im,cnts,heir = cv2.findContours(threshold.copy(),cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
        if (len(cnts) > 0):
            c = max(cnts,key = cv2.contourArea)
            ((x_circle,y_circle),self.cntR) = cv2.minEnclosingCircle(c)

            M = cv2.moments(c)
            if (M['m00'] > 0):
                self.x = int(M['m10']/M['m00'])
                self.y = int(M['m01']/M['m00'])
        return (self.cntR**2 * math.pi)
```

7.2 Inverse Kinematics

Given a target orientation, the necessary servo angles had to be computed. The inverse kinematics solution is implemented below,

```python
import numpy as np

class InverseKinematicsConverter:


    def __init__(self,T, psi, phi, theta):
        self.psi = np.deg2rad(psi)
        self.phi = np.deg2rad(phi)
        self.theta = np.deg2rad(theta)
        self.T = T

        self.rb = np.array([[np.cos(self.psi) * np.cos(self.theta),
        -np.sin(self.psi) * np.cos(self.phi) + np.cos(self.psi) * np.sin(self.theta) * np.sin(self.phi),
        np.sin(self.psi) * np.sin(self.phi) + np.cos(self.psi) * np.sin(self.theta) * np.cos(self.phi)],
        [np.sin(self.psi)*np.cos(self.theta),
        np.cos(self.psi)*np.cos(self.phi) + np.sin(self.psi) * np.sin(self.theta) * np.sin(self.phi),
        -np.cos(self.psi) * np.sin(self.phi) + np.sin(self.psi) * np.sin(self.theta) * np.cos(self.phi)],
        [-np.sin(self.theta),
        np.cos(self.theta) * np.sin(self.phi),
        np.cos(self.theta) * np.cos(self.phi)
        ]])


        self.a_1 = 4.41
        self.b_1 = 0.76
        self.a_2 = 4.67
        self.b_2 = 1.55
        self.la = 6
        self.lb = 0.4

    def calcP(self):
        self.p1 = np.transpose(np.array([self.b_1,-self.a_1 * np.sqrt(3)/2,0]))
        # print('p1',self.p1)
        self.p2 = np.transpose(np.array([self.a_1 - np.cos(np.deg2rad(60)) * (self.a_1/2-self.b_1),
        (self.a_1/2 - self.b_1) * np.sin(np.deg2rad(60)),0]))
        # print('p2',self.p2)
        self.p3 = np.transpose(np.array([self.a_1 - np.cos(np.deg2rad(60)) * (self.a_1/2+self.b_1),
        (self.a_1/2 + self.b_1) * np.sin(np.deg2rad(60)),0]))
        # print('p3',self.p3)
        self.p4 = np.transpose(np.array([-self.a_1 + np.cos(np.deg2rad(60)) * (self.a_1/2 + self.b_1),
        (self.a_1/2 + self.b_1) * np.sin(np.deg2rad(60)),0]))
        # print('p4',self.p4)
        self.p5 = np.transpose(np.array([-self.a_1 + np.cos(np.deg2rad(60)) * (self.a_1/2 - self.b_1),
        (self.a_1/2 - self.b_1) * np.sin(np.deg2rad(60)),0]))
        # print('p5',self.p5)
        self.p6 = np.transpose(np.array([-self.b_1,-self.a_1 * np.sqrt(3)/2,0]))
        # print('p6',self.p6)

    def calcB(self):
        self.b1 = np.array([self.a_2 - np.cos(np.deg2rad(60)) * (self.a_2/2 + self.b_2),
        -np.sin(np.deg2rad(60)) * (self.a_2/2 + self.b_2),0])
        # print('b1',self.b1)
        self.b2 = np.array([self.a_2 - np.cos(np.deg2rad(60)) * (self.a_2/2 - self.b_2),
        -np.sin(np.deg2rad(60)) * (self.a_2/2 - self.b_2),0])
        # print('b2',self.b2)
        self.b3 = np.array([self.b_2,self.a_2 * np.sqrt(3)/2,0])
        # print('b3',self.b3)
        self.b4 = np.array([-self.b_2,self.a_2 * np.sqrt(3)/2,0])
        # print('b4',self.b4)
        self.b5 = np.array([-self.a_2 + np.cos(np.deg2rad(60)) * (self.a_2/2 - self.b_2),
        -np.sin(np.deg2rad(60)) * (self.a_2/2 - self.b_2),0])
        # print('b5',self.b5)
        self.b6 = np.array([-self.a_2 + np.cos(np.deg2rad(60)) * (self.a_2/2 + self.b_2),
        -np.sin(np.deg2rad(60)) * (self.a_2/2 + self.b_2),0])
        # print('b6',self.b6)

    def calcK(self):
        self.k1 = self.T + self.rb @ self.p1 - self.b1
        self.k2 = self.T + self.rb @ self.p2 - self.b2
        self.k3 = self.T + self.rb @ self.p3 - self.b3
        self.k4 = self.T + self.rb @ self.p4 - self.b4
        self.k5 = self.T + self.rb @ self.p5 - self.b5
        self.k6 = self.T + self.rb @ self.p6 - self.b6

        self.k_1 = np.linalg.norm(self.k1)
        self.k_2 = np.linalg.norm(self.k2)
        self.k_3 = np.linalg.norm(self.k3)
        self.k_4 = np.linalg.norm(self.k4)
        self.k_5 = np.linalg.norm(self.k5)
        self.k_6 = np.linalg.norm(self.k6)

        # print('k1',self.k_1)
        # print('k2',self.k_2)
        # print('k3',self.k_3)
        # print('k4',self.k_4)
        # print('k5',self.k_5)
        # print('k6',self.k_6)


    def calcAngles(self):
        self.calcP()
        self.calcB()
        self.calcK()

        cons1 = (self.la**2 - self.k_1**2 - self.lb**2)/-2
        cons2 = (self.la**2 - self.k_2**2 - self.lb**2)/-2
        cons3 = (self.la**2 - self.k_3**2 - self.lb**2)/-2
        cons4 = (self.la**2 - self.k_4**2 - self.lb**2)/-2
        cons5 = (self.la**2 - self.k_5**2 - self.lb**2)/-2
        cons6 = (self.la**2 - self.k_6**2 - self.lb**2)/-2

        xvec = np.linspace(-np.pi/2,np.pi/2,100)

        lb12x = self.lb * np.cos(xvec) * np.cos(np.deg2rad(60))
        lb12y = self.lb * np.cos(xvec) * np.sin(np.deg2rad(60))
```

```python
        lb56x = self.lb * np.cos(xvec) * np.cos(np.deg2rad(60))
        lb56y = -self.lb * np.cos(xvec) * np.sin(np.deg2rad(60))
        lb34x = -self.lb * np.cos(xvec) * np.sin(np.deg2rad(60))
        lb34y = 0 * xvec


        lbz = self.lb * np.sin(xvec)
        leg1 = self.k1[0] * lb12x + self.k1[1] * lb12y + self.k1[2] * lbz
        leg2 = self.k2[0] * lb12x + self.k2[1] * lb12y + self.k2[2] * lbz
        leg3 = self.k3[0] * lb34x + self.k3[1] * lb34y + self.k3[2] * lbz
        leg4 = self.k4[0] * lb34x + self.k4[1] * lb34y + self.k4[2] * lbz
        leg5 = self.k5[0] * lb56x + self.k5[1] * lb56y + self.k5[2] * lbz
        leg6 = self.k6[0] * lb56x + self.k6[1] * lb56y + self.k6[2] * lbz

        idx1 = np.argmin(np.abs(leg1-cons1))
        idx2 = np.argmin(np.abs(leg2-cons2))
        idx3 = np.argmin(np.abs(leg3-cons3))
        idx4 = np.argmin(np.abs(leg4-cons4))
        idx5 = np.argmin(np.abs(leg5-cons5))
        idx6 = np.argmin(np.abs(leg6-cons6))

        beta1 = xvec[idx1]
        beta2 = xvec[idx2]
        beta3 = xvec[idx3]
        beta4 = xvec[idx4]
        beta5 = xvec[idx5]
        beta6 = xvec[idx6]


        ang1 = np.rad2deg(beta1 + np.pi/2 - np.deg2rad(6))*(180/156)
        ang2 = (np.rad2deg(beta2 + np.pi/2) - 7) * (180/164)
        ang3 = (np.rad2deg(beta3 + np.pi/2) - 2)  * (180/158)
        ang4 = np.rad2deg(beta4 + np.pi/2) * (180/160)
        ang5 = (np.rad2deg(beta5 + np.pi/2) - 6) * (180/160)
        ang6 = (np.rad2deg(beta6 + np.pi/2) - 2) * (180/163)

        lis = [ang1,ang2,ang3,ang4,ang5,ang6]
        for idx,ang in enumerate(lis):
            if ang <= 0:
                lis[idx] = 0
            elif ang > 180:
                lis[idx] = 180

        return lis

    def updateT(self,T):
        self.T = T

    def updatePsi(self,psi):
        self.psi = np.deg2rad(psi)

        self.rb = np.array([[np.cos(self.psi) * np.cos(self.theta),
        -np.sin(self.psi) * np.cos(self.phi) + np.cos(self.psi) * np.sin(self.theta) * np.sin(self.phi),
        np.sin(self.psi) * np.sin(self.phi) + np.cos(self.psi) * np.sin(self.theta) * np.cos(self.phi)],
        [np.sin(self.psi)*np.cos(self.theta),
        np.cos(self.psi)*np.cos(self.phi) + np.sin(self.psi) * np.sin(self.theta) * np.sin(self.phi),
        -np.cos(self.psi) * np.sin(self.phi) + np.sin(self.psi) * np.sin(self.theta) * np.cos(self.phi)],
        [-np.sin(self.theta),
        np.cos(self.theta) * np.sin(self.phi),
        np.cos(self.theta) * np.cos(self.phi)
        ]])


    def updatePhi(self,phi):
        self.phi = np.deg2rad(phi)


        self.rb = np.array([[np.cos(self.psi) * np.cos(self.theta),
        -np.sin(self.psi) * np.cos(self.phi) + np.cos(self.psi) * np.sin(self.theta) * np.sin(self.phi),
        np.sin(self.psi) * np.sin(self.phi) + np.cos(self.psi) * np.sin(self.theta) * np.cos(self.phi)],
        [np.sin(self.psi)*np.cos(self.theta),
        np.cos(self.psi)*np.cos(self.phi) + np.sin(self.psi) * np.sin(self.theta) * np.sin(self.phi),
        -np.cos(self.psi) * np.sin(self.phi) + np.sin(self.psi) * np.sin(self.theta) * np.cos(self.phi)],
        [-np.sin(self.theta),
        np.cos(self.theta) * np.sin(self.phi),
        np.cos(self.theta) * np.cos(self.phi)
        ]])

    def updateTheta(self,theta):
        self.theta = np.deg2rad(theta)


        self.rb = np.array([[np.cos(self.psi) * np.cos(self.theta),
        -np.sin(self.psi) * np.cos(self.phi) + np.cos(self.psi) * np.sin(self.theta) * np.sin(self.phi),
        np.sin(self.psi) * np.sin(self.phi) + np.cos(self.psi) * np.sin(self.theta) * np.cos(self.phi)],
        [np.sin(self.psi)*np.cos(self.theta),
        np.cos(self.psi)*np.cos(self.phi) + np.sin(self.psi) * np.sin(self.theta) * np.sin(self.phi),
        -np.cos(self.psi) * np.sin(self.phi) + np.sin(self.psi) * np.sin(self.theta) * np.cos(self.phi)],
        [-np.sin(self.theta),
        np.cos(self.theta) * np.sin(self.phi),
        np.cos(self.theta) * np.cos(self.phi)
        ]])



if __name__ == '__main__':
    T = np.array([0,0,5.8])
    T = np.transpose(T)
    phi = 0
    psi = 0
    theta = 3
    converter = InverseKinematicsConverter(T,psi,phi,theta)
    angles = converter.calcAngles()
    print(angles)
```

WBS NUMBER	TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION	PHASE ONE			PHASE TWO			PHASE THREE			PHASE FOUR															
																		WEEK 11					WEEK 11									
						M	M	M	M	M	M	M	M	M	M	M	M	M	T	W	Th	F	M	T	W	Th	F	M	T	W	Th	F
1	**Project Conception and Initiation**					**8/27**	**9/3**	**9/10**	**9/17**	**9/24**	**10/1**	**10/8**	**10/15**	**10/22**	**10/29**	**11/5**	**11/12**	**11/19**					**11/26**					**12/3**				
1.1	Create Conceptual Design	Dennis	8/26/18	9/3/18	7	🟦	🟦																									
1.1	Create CAD design	Dennis, Michael	9/5/18	9/19/18	14		🟦	🟦	🟦																							
1.1.1	Purchase/Acquire Hardware	Rob, Michael	9/5/18	9/19/18	14		🟦	🟦	🟦																							
1.2	Solve Inverse Kinematics Problem for platform	Dennis	8/26/18	8/29/18	3	🟦	🟦	🟦	🟦	🟦																						
2	**Project Execution**																															
2.1	Write Closed-Loop Control Code	Kaustubh	10/22/18	11/19/18	27									🟦	🟦	🟦	🟦	🟦														
2.2	Manufacture platform	Dennis, Michael, Rob	9/19/18	10/17/18	28				🟦	🟦	🟦	🟦	🟦																			
2.2.1	Write Servo Control Code	Dennis	10/3/18	10/31/18	28						🟦	🟦	🟦	🟦	🟦																	
2.2.2	Test Open-Loop Control	Dennis	10/31/18	11/14/18	14										🟦	🟦	🟦															
2.2.3	Test and Troubleshoot Closed-Loop Control	Kaustubh, Dennis	11/19/18	11/29/18	10													🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦							
3	**Data Analysis and Evaluation**																															
3.1	Collect data on repeatability, accuracy, etc.	Kaustubh, Dennis	11/14/18	11/29/18	15													🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦							
3.2	Analyze data	Dennis, Michael, Kaustubh	11/17/18	12/3/18	16													🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦				
3.3	Create Final Presentation and Report	All	12/2/18	12/7/18	5																					🟦	🟦	🟦	🟦	🟦	🟦	🟦