NYU TANDON SCHOOL OF ENGINEERING

Advanced Mechatronics Project Two Report

3RPS + Gantry System for Motor Rehabilitation

Ashwik Tatineni
Josephine Odusanya
Vedvyas Danturi

NYU | TANDON SCHOOL OF ENGINEERING

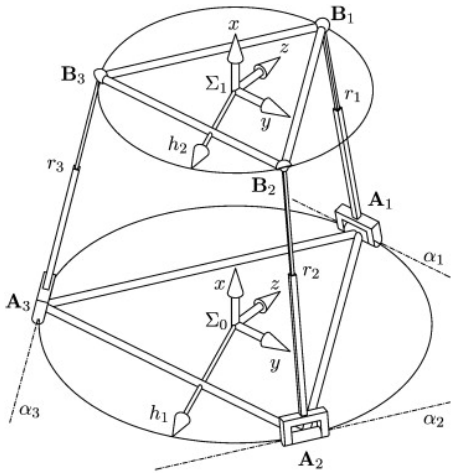
Abstract

Stroke rehabilitation often requires specialized tools to aid in balance and gait recovery. This project improves upon the previously explored **stewart platform**. Due to **limited workspace** and high potential for **singularities** in the translation of the leg, we decided to **separate out** the cartesian 2-d degree of freedom into a mechanism and the leg height + angle being the other mechanism and came up with the **3RPS fixed on a 2D- Cartesian gantry**. This allows us to modify these parameters independentantly. Due to the requirement of continuous motion in **multiple servos simultaneously**, we use the Propeller activity board. We give 2 joysticks for the control of the system. While also providing the IMU values to adjust the pitch of the platform which would affect the leg's orientation. **Future** work will focus on making this **spline based control** and creating **data structures** around it, and loading the complete **spline on-board using the SD-card** itself.

Keywords : Parallax Propeller, Gait rehabilitation, IMU, Robotics, Stroke rehabilitation

NYU TANDON SCHOOL OF ENGINEERING

1 Introduction

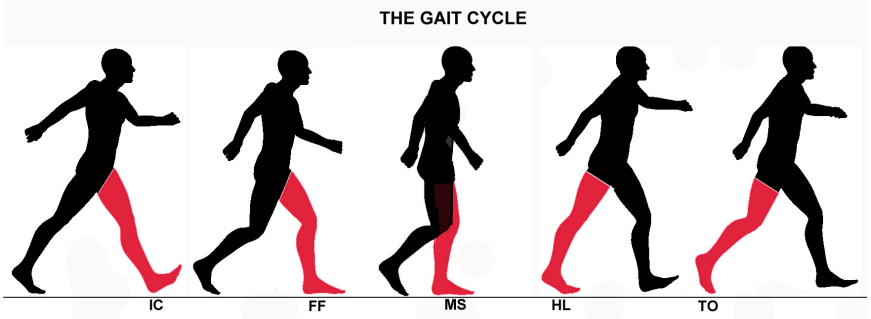


The 3RPS platform is a type of parallel manipulator or parallel robotic platform that consists of three **Revolute-Prismatic-Spherical (RPS)** kinematic chains connecting a fixed base to a moving platform. It is widely used in robotics, precision positioning, and motion simulation due to its stability, rigidity, and ease of kinetic chain structure.

This Platform is added on a **2D - Cartesian gantry** which gives the platform it's translation motion, these systems together create a highly adaptable **5-DOF platform**, which is beneficial for our gait rehabilitation research but could be adapted into any type of task by being **modular** and having **mounting holes** to attach at all DoF's end points.

1.1 Problem Statement

Stroke is a leading cause of long-term disability, often resulting in impaired mobility and balance. Rehabilitation plays a crucial role in restoring motor function, with a primary focus on gait training and postural stability. Traditional rehabilitation methods rely on physical therapy and mechanical aids; however, advancements in robotics and automation have introduced new possibilities for more precise and adaptive rehabilitation tools.



By leveraging its ability to make controlled, multi-axis movements, our novel 5-DoF platform can be adapted for rehabilitation purposes, particularly for balance training and gait assessment in stroke patients.

1.2 Related Work

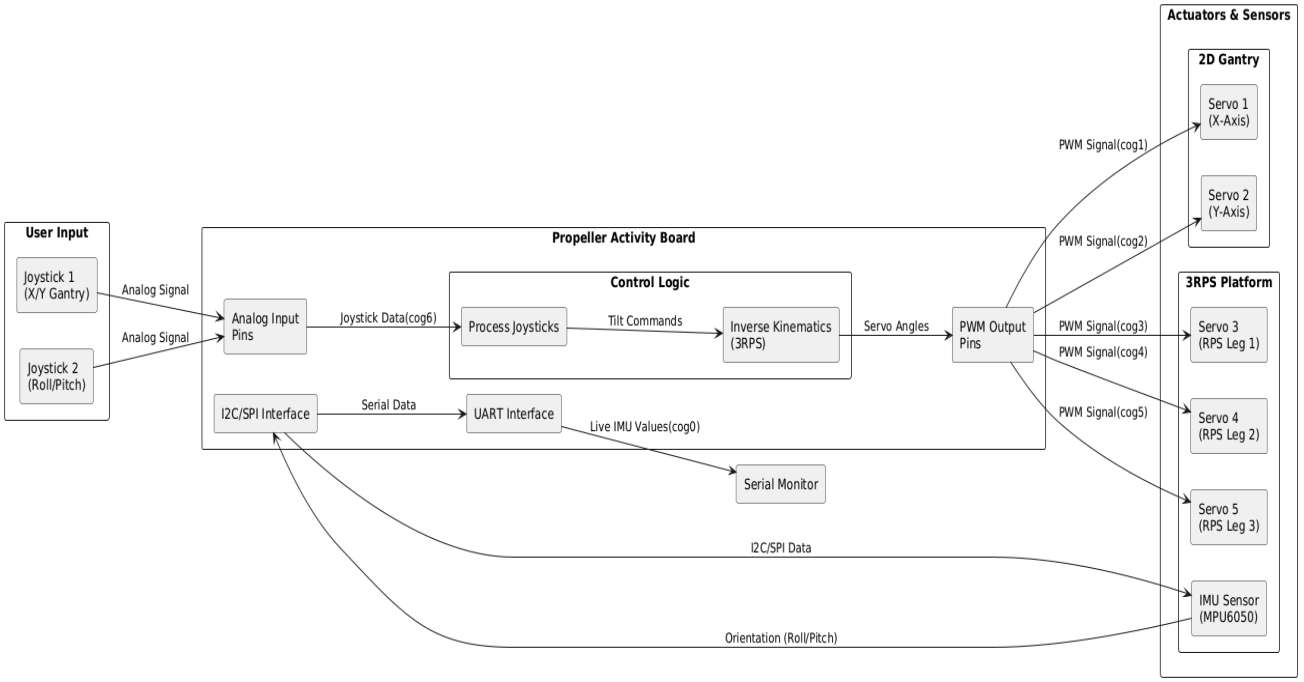
A Low-Cost Stewart Platform For Motion Study in Surgical Robotics [1]. In this document, researchers present a miniaturized Stewart Platform (SPRK) designed to simulate organ motion for surgical robotics research. It fits within a da Vinci Research Kit workspace and provides ±1.27 cm translation and ±15° rotation. The platform supports various motion modes and achieves high positional accuracy for experimental applications.

A Stewart Platform-Based System for Ankle Telerehabilitation[3] In this document, researchers explore the Rutgers Ankle is a Stewart platform-based haptic interface designed for orthopedic rehabilitation. It provides six-DOF resistive forces for virtual reality-based exercises, with movement and force data recorded for remote monitoring. Initial trials showed promising results as a diagnostic and rehabilitation tool, though further medical validation is needed.

In the “Dynamic Analysis of a Stewart Platform for Lower Human Limb Rehabilitation Using Cad Tools [4], the article discusses the design and analysis of a Stewart platform prototype for rehabilitating children with cerebral palsy, using servo linear actuators. Kinetic analysis through SolidWorks provided actuator speed profiles and force requirements to ensure proper movement. The platform aims to assist in the motor rehabilitation of children by simulating walking motions and improving mobility for those with locomotion problems.

In “Gantry Systems: Working Outside the Envelope”, the article highlights the advantages of gantry (Cartesian) robots over traditional articulated and SCARA arms, especially in terms of workspace efficiency and scalability. Gantry robots offer high accuracy, low cost, and ease of programming, making them ideal for tasks like palletizing, pick and place, and packaging. Their rigid structure and ability to utilize nearly the entire work envelope make them a versatile and often overlooked solution in industrial automation[6].

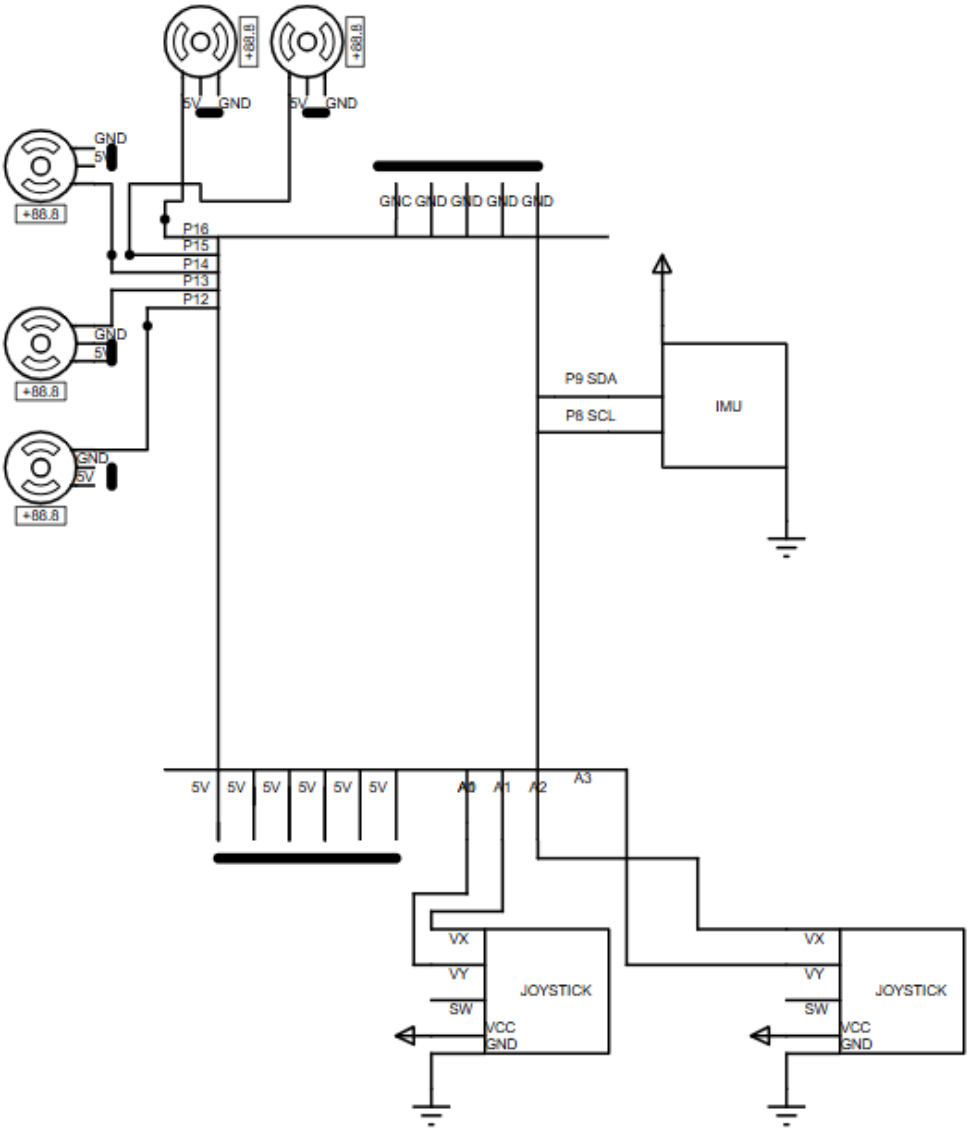
2 Design and Working Principle



The above is a block diagram of how the system is meant to work.

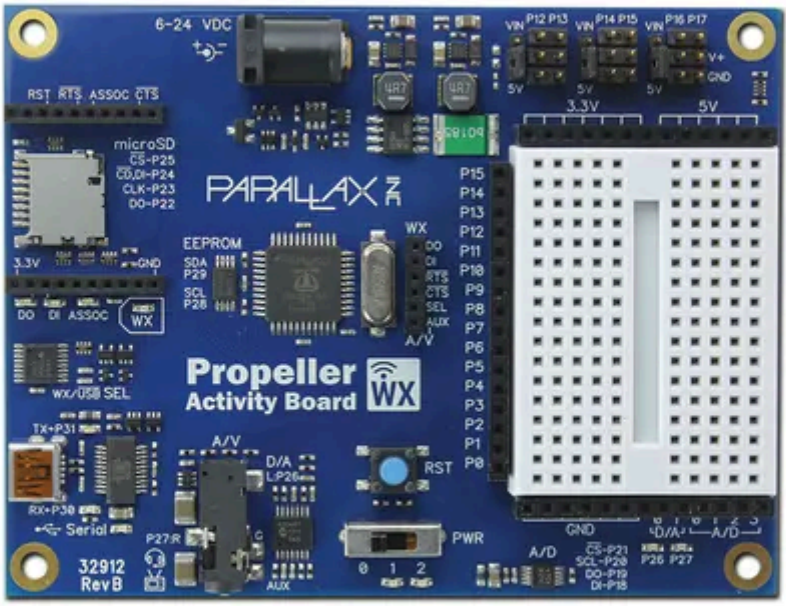
The Propeller AB acts as the brain power for the entire mechanism. It will process the data sent from the IMU via I2C. The orientation obtained from this will be used to determine the PWM speeds that will control each motor in order to change its position. Currently only implementing simple speed control of the various continuous servos using pulseout command. The joystick tells us to go in which direction and the user can start and stop according to the values displayed by the IMU. The logic used is simple setting up flags for the joystick value and using that to determine the status of how to run the servos.

3 Hardware Selection and Integration

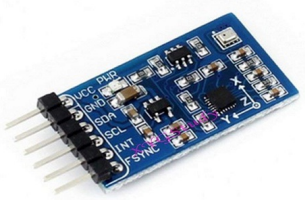


To achieve seamless operation, careful hardware selection and integration are essential. The following components are incorporated into the system:

1. **Microcontroller:** Propeller Activity Board, handling IMU data, joystick input and servo control.

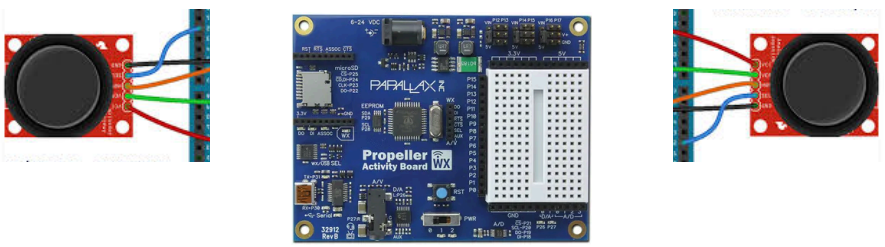


2. **Sensors**: Inertial Measurement Unit (IMU) for real-time orientation tracking.



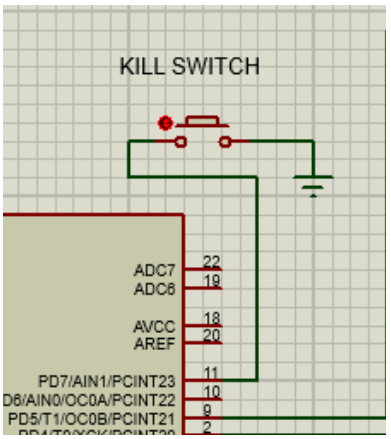
3. **Actuators:** Five Servo motors enabling precise platform adjustments and using activity board servo headers.

4. **Joystick**: the addition of the joy-stick to control the top panel into a desired pose.



3.1 Safety Mechanism

Due to the use of many DC motors, we chose to implement a simple kill switch controlled by an interrupt service.



4 Control Analysis

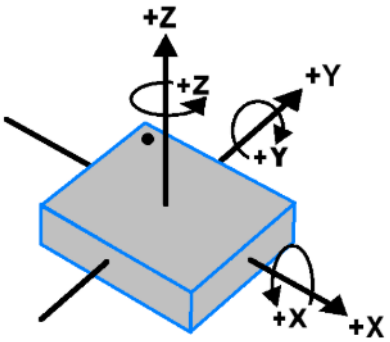
4.1 Forward Kinematics

The forward kinematics of our 5DoF 3RPS + Gantry platform involves determining the position and orientation of the moving platform given the lengths of the all traversed rack and pinion modules. This problem is simple due to the parallel nature of the mechanism acting in accordance with the DoF's control.

1. Cog 0: Reads joysticks and sets threshold flags (LEFT, RIGHT, UP, DOWN) and height/pitch targets.
2. Cogs 1–2: Dedicated to gantry servos (X/Y), reacting to flags in while loops.
3. Cogs 3–5: Control 3RPS platform servos, adjusting based on shared target_height and target_pitch.
4. Cog 6: Continuously reads IMU data and sends it to the serial monitor.
5. Cog 7: Monitors servo positions and IMU data for safety limits, triggering emergency stops.
6. Shared Variables: Static Volatile flags used for inter-cog communication (flags, targets, angles).

4.2 IMU (Inertial Measurement Unit)

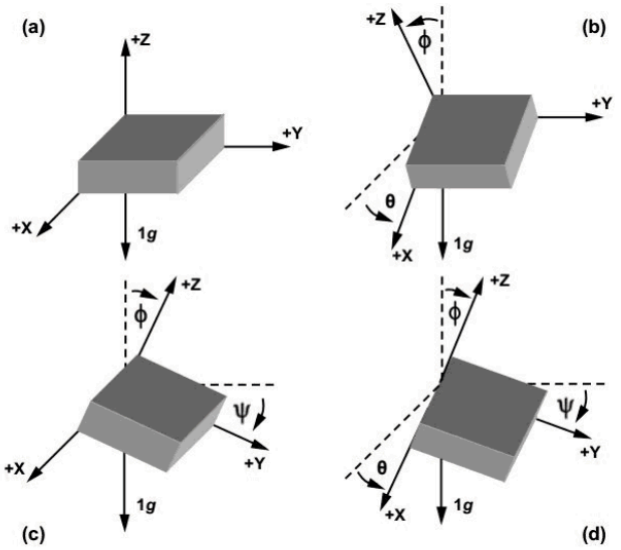
The MPU 6050 sensor device combines a gyroscope, accelerometer and digital motion processor in a compact module.



MPU-6050
Orientation & Polarity of Rotation

When the gyroscopes are rotated about any of the sense axes, the Coriolis Effect causes a vibration that is detected by a MEM inside MPU6050. The resulting signal is amplified, demodulated, and filtered to produce a voltage that is proportional to the angular rate. This voltage is digitized using 16-bit ADC to sample each axis. The full-scale range of output are +/- 250, +/- 500, +/- 1000, +/- 2000. It measures the angular velocity along each axis in degree per second unit

In comparison, the 3-axis accelerometer consists of Micro Electro Mechanical (MEMs) technology. It is used to detect angle of tilt or inclination along the X, Y and Z axes as shown in below figure.



The full-scale range of acceleration are +/- 2g, +/- 4g, +/- 8g, +/- 16g and it is measured in g (gravity force) units. When a device is placed on a flat surface it will measure 0g on the X and Y axis and +1g on the Z axis.

In this project, we will only take readings from the gyroscope to determine the angle of tilt/inclination.

The math behind the IMU is **Kalman Filtering**.The Kalman Filter operates through a recursive two-step process: Prediction and Update.

1. Prediction Step:

In this step, the filter predicts the future state of the system (i.e., the ball's position and velocity) based on the previous state estimate and a dynamic model of the system. This is achieved using the state transition matrix, Fk.

The prediction is represented mathematically as:

$$\hat{x}_k = F_k * \hat{x}_{k-1} + B_k * u_k$$

$$P_k = F_k * P_{k-1} * FT_k + Q_k$$

Where:

x̂k is the predicted state at time step k

Fk is the state transition matrix

x̂k-1 is the estimated state at time step k-1

Bk is the control matrix (if any external forces are present)

uk is a control vector

Pk is the predicted covariance matrix

Pk-1 is the covariance matrix at time step k-1

FTk is the transpose of the state transition matrix

Qk is the process noise covariance matrix

The filter also propagates the uncertainty (covariance) associated with the previous state estimate through the prediction step. This is done using the process noise covariance matrix, Qk.

2. Update Step (Measurement Correction):

Here the filter receives a measurement from the camera (zk) representing the actual position of the ball. It then calculates a Kalman gain (K), which determines how much weight to give to the measurement versus the prediction. The Kalman Gain is expressed as:

$$\mathbf{K = Pk * HTk * (Hk * Pk * HTk + Rk)^{-1}}$$

Where:

K is the Kalman gain matrix

Hk is the measurement matrix, which relates the state to the measurement.

Rk is the measurement noise covariance matrix

It updates the state estimate based on the difference between the predicted measurement and the actual measurement, weighted by the Kalman gain.

$$\hat{x}'k = \hat{x}k + K * (zk - Hk * \hat{x}k)$$

Similarly, it updates the uncertainty (covariance) based on the Kalman gain.

$$P'k = Pk - K * Hk * Pk$$

Where:

P'k is the updated covariance matrix

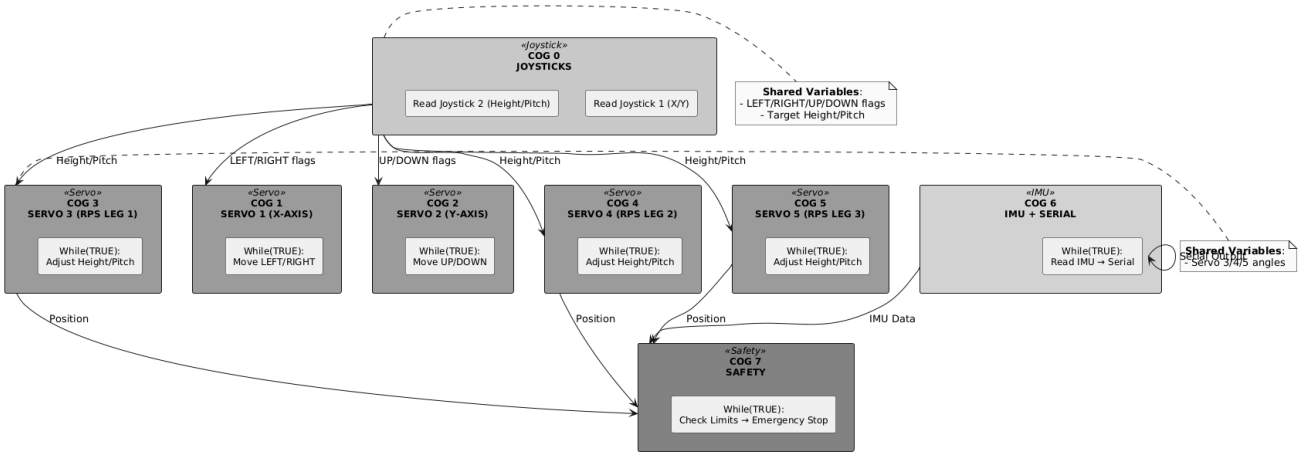
x̂'k is the updated state

Applying a Kalman Filter in this system provides several benefits:

1. The Kalman Filter smooths out the noisy camera measurements, resulting in a more stable estimate of the ball's position.
2. By combining predictions with measurements, the filter allows the system to respond more quickly to changes in the ball's position.

5 Software Design (flow logic)

This flowchart visually represents the software logic for controlling the platform using joystick data and servo control. It starts with initializing the joysticks, IMU and motors, followed by continuously reading roll and pitch values. These values are mapped to motor PWM signals to adjust the platform's tilt. The system then applies the calculated servo speeds and prints debugging data before looping back to read new IMU data, ensuring real-time balance adjustments.



Several important libraries were implemented to make this control possible such as the imu_sensor.h which we created and customised ourselves.

6 System Overview (Prototype)

The following is a model showing the final design of the Stewart Platform.

Model of 3RPS+Gantry Platform- Figure 1

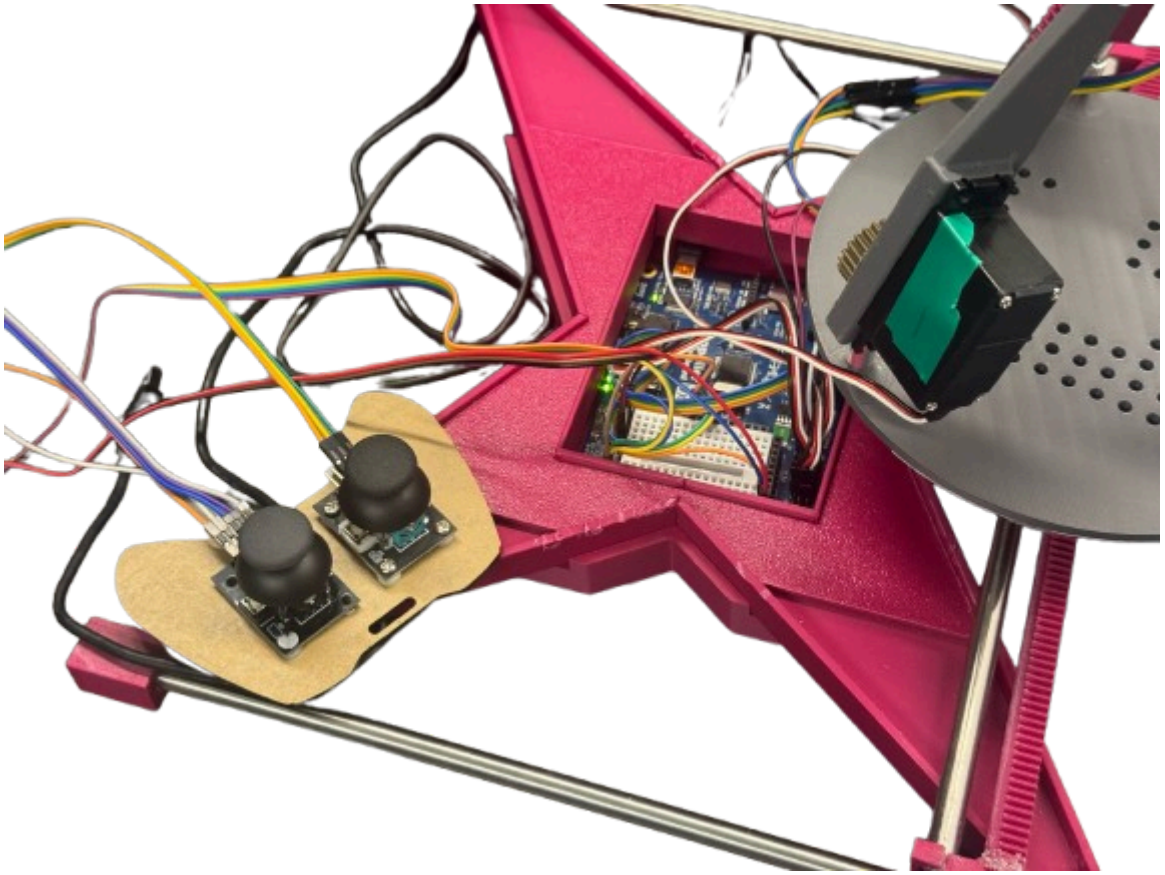


Figure 2

6.1 Bill of materials

Materials	Cost (USD)
Propeller Activity Board	76
IMU	10
Parallax Continuous Servo x 6	16x6 = 96
Linear bearings + rods x 3	36
Legs (3D printed)	free.99
Universal Joints	7
Total	**225**

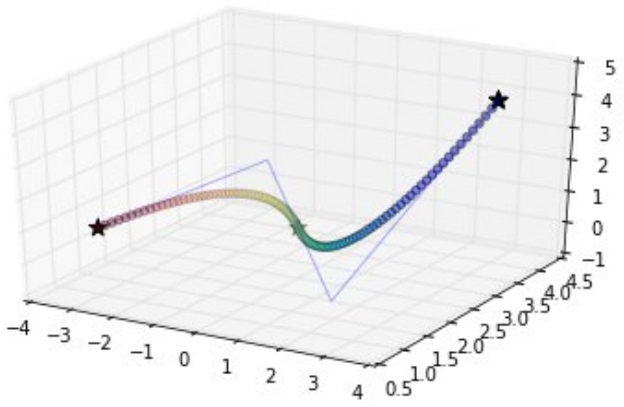
6.2 Results

With this simple design of our 5DoF Platform, we were able to prove proof of concept in preparation for the third stage of the project. Previous issues we encountered had mainly to do with the linear actuators we made and the motors used which gave a low torque compared to what we wanted. So after three interactions, our lead screw mechanism broke and we could not put the whole prototype together. Our circuit was completed though, which has given us proof of concept despite the print design flaw. This new design utilizing rack and pinion with continuous servos made the implementation much easier and we kept the design barebones to not have any type of friction hindering the movement, while also using linear bearings on 8mm rods to provide rigidity to the structure. This change enabled us to successfully put together the prototype above and present a live demo of its function.

7 Future development and Conclusion

Project 3 will improve upon traditional Stewart platform designs to create a more adaptable rehabilitation tool focused on balance and gait recovery for stroke patients. To address workspace constraints and singularity issues during leg translation, the system separates movement into two subsystems: a 2D Cartesian gantry for planar motion and a 3RPS mechanism for controlling leg height and orientation. This modular approach allows independent control over translation and articulation, enabling more natural and customizable movement patterns.

The system is controlled entirely using a Raspberry Pi, which handles real-time servo coordination and collects IMU feedback to assist in leg orientation control. We will add spline recording from synchronized camera and IMU data on subjects, and develop supporting data structures to improve motion control and repeatability. These splines will be stored and loaded from an onboard SD card, enabling flexible, programmable movement tailored to individual rehabilitation needs.



8 References

1. V. Patel, S. Krishnan, A. Goncalves and K. Goldberg, "SPRK: A low-cost stewart platform for motion study in surgical robotics," 2018 International Symposium on Medical Robotics (ISMR), Atlanta, GA, USA, 2018, pp. 1-6, doi: 10.1109/ISMR.2018.8333300.
2. Maclachlan RA, Becker BC, Tabarés JC, Podnar GW, Lobes LA Jr, Riviere CN. Micron: an Actively Stabilized Handheld Tool for Microsurgery. IEEE Trans Robot. 2012 Feb 1;28(1):195-212. doi: 10.1109/TRO.2011.2169634. Epub 2011 Nov 18. PMID: 23028266; PMCID: PMC3459596.
3. Girone, M., Burdea, G., Bouzit, M. *et al.* A Stewart Platform-Based System for Ankle Telerehabilitation. *Autonomous Robots* 10, 203–212 (2001). https://doi.org/10.1023/A:1008938121020
4. “Dynamic Analysis of a Stewart Platform for Lower Human Limb Rehabilitation Using Cad Tools”, G. HoffmanF.Ramirez, 2018, https://api.semanticscholar.org/CorpusID:221169673
5. https://www.electronicwings.com/sensors-modules/mpu6050-gyroscope-accelerometer-temperature-sensor-module
6. https://www.macrondynamics.com/job-stories/gantry-systems-overview/

9 Appendix

IMU Library

i) IMU Header file

```
/**
 * @file include/imu_sensor.h
 * @brief Provides IMU 6050 specific functions
 *
 * Copyright (c) 2025 by me (Josephine Odusanya)
 */

#ifndef IMU_SENSOR_H
#define IMU_SENSOR_H

#ifdef __cplusplus
extern "C"
{
#endif

#include "cog.h"
#include <stdint.h>
#include <string.h>
#include "simpletools.h"
#include "simplei2c.h"
#include "math.h"

i2c *imu;  // I2C bus for MPU-60X0

#define MPU6050_ADDR 0x68  // Default I2C address (AD0 pin low)

// Register addresses from the datasheet
#define PWR_MGMT_1     0x6B
#define GYRO_CONFIG    0x1B
#define ACCEL_CONFIG   0x1C
#define ACCEL_XOUT_H   0x3B
#define TEMP_OUT_H     0x41
#define GYRO_XOUT_H    0x43

// Sensor data structure
typedef struct {
    int16_t accel[3];  // X,Y,Z
    int16_t gyro[3];   // X,Y,Z
```

```
    int16_t temp;
} imu_data_t;


HUBTEXT void mpu6050_init();

HUBTEXT void mpu6050_read(imu_data_t *data);

HUBTEXT void calibrate_gyro(volatile int samples,volatile int16_t
*offsets);

HUBTEXT void current_pos();


#endif  // IMU_SENSOR_H


#ifdef __cplusplus
}
#endif
```

ii) IMU C file

```
#include "simpletools.h"
#include "simplei2c.h"
#include "math.h"

#define MPU6050_ADDR 0x68  // Default I2C address (AD0 pin low)
#define SCL_PIN 8          // Use P0 for SCL
#define SDA_PIN 9          // Use P1 for SDA

i2c *imu;  // I2C bus for MPU-60X0

// Register addresses from the datasheet
#define PWR_MGMT_1     0x6B
#define GYRO_CONFIG    0x1B
#define ACCEL_CONFIG   0x1C
#define ACCEL_XOUT_H   0x3B
#define TEMP_OUT_H     0x41
#define GYRO_XOUT_H    0x43

// Sensor data structure
```

```
typedef struct {
    int16_t accel[3];  // X,Y,Z
    int16_t gyro[3];   // X,Y,Z
    int16_t temp;
} imu_data_t;

volatile int samples;
volatile int16_t *offsets;
volatile int16_t gyro_offsets[3];

// Initialize MPU-60X0
void mpu6050_init() {
    // Initialize I2C
    i2c_open(&imu, SCL_PIN, SDA_PIN, 0);

    // Wake up MPU-60X0 (disable sleep mode)
    i2c_start(&imu);
    i2c_writeByte(&imu, MPU6050_ADDR << 1);
    i2c_writeByte(&imu, PWR_MGMT_1);
    i2c_writeByte(&imu, 0x00);  // Wake up (clear sleep bit)
    i2c_stop(&imu);

    // Configure gyroscope range (±250°/s)
    i2c_start(&imu);
    i2c_writeByte(&imu, MPU6050_ADDR << 1);
    i2c_writeByte(&imu, GYRO_CONFIG);
    i2c_writeByte(&imu, 0x00);  // ±250°/s (FS_SEL=0)
    i2c_stop(&imu);

    // Configure accelerometer range (±2g)
    i2c_start(&imu);
    i2c_writeByte(&imu, MPU6050_ADDR << 1);
    i2c_writeByte(&imu, ACCEL_CONFIG);
    i2c_writeByte(&imu, 0x00);  // ±2g (AFS_SEL=0)
    i2c_stop(&imu);

    // Optional: Configure DLPF (Digital Low Pass Filter) //NOTE FOR
ME ADD A KFILTER HERE
    // i2c_start(&imu);
    // i2c_writeByte(&imu, MPU6050_ADDR << 1);
    // i2c_writeByte(&imu, 0x1A);  // CONFIG register
    // i2c_writeByte(&imu, 0x03);  // DLPF_CFG=3 (44Hz accel, 42Hz
gyro)
```

```
    // i2c_stop(&imu);
}

// Read all sensor data from MPU-60X0
void mpu6050_read(imu_data_t *data) {
    uint8_t buf[14];

    // Start reading from register 0x3B (ACCEL_XOUT_H)
    i2c_start(&imu);
    i2c_writeByte(&imu, MPU6050_ADDR << 1);
    i2c_writeByte(&imu, ACCEL_XOUT_H);

    // Repeated start to begin reading
    i2c_start(&imu);
    i2c_writeByte(&imu, (MPU6050_ADDR << 1) | 1);

    // Read 14 bytes (accel, temp, gyro)
    for(int i = 0; i < 13; i++) {

        buf[i] = i2c_readByte(&imu, 0);  // ACK all but last byte

    }
    buf[13] = i2c_readByte(&imu, 1);     // NAK last byte
    i2c_stop(&imu);

    // Format data (registers are big-endian)
    data->accel[0] = (buf[0] << 8) | buf[1];   // X
    data->accel[1] = (buf[2] << 8) | buf[3];   // Y
    data->accel[2] = (buf[4] << 8) | buf[5];   // Z
    data->temp     = (buf[6] << 8) | buf[7];   // Temperature
    data->gyro[0]  = (buf[8] << 8) | buf[9];   // X
    data->gyro[1]  = (buf[10] << 8) | buf[11]; // Y
    data->gyro[2]  = (buf[12] << 8) | buf[13]; // Z

}

// Calibrate gyroscope by averaging samples while stationary
void calibrate_gyro(volatile int samples, volatile int16_t *offsets)
{
    imu_data_t data;
    int32_t sum[3] = {0};
```

```
    void calibrate_gyro(volatile int samples, volatile int16_t
*offsets);
    print("Calibrating gyro... keep sensor still!\n");

    for(int i = 0; i < samples; i++) {
        mpu6050_read(&data);
        sum[0] += data.gyro[0];
        sum[1] += data.gyro[1];
        sum[2] += data.gyro[2];
        pause(10);
    }

    offsets[0] = sum[0] / samples;
    offsets[1] = sum[1] / samples;
    offsets[2] = sum[2] / samples;

    print("Offsets: X=%d, Y=%d, Z=%d\n", offsets[0], offsets[1],
offsets[2]);

}

void current_pos(void) {

    // Main loop
        imu_data_t data;
        mpu6050_read(&data);

        // Apply gyro calibration
        data.gyro[0] -= gyro_offsets[0];
        data.gyro[1] -= gyro_offsets[1];
        data.gyro[2] -= gyro_offsets[2];

        // Convert raw data to human-readable values:
        // Accelerometer: ±2g range (16384 LSB/g)
        float ax = data.accel[0] / 16384.0;
        float ay = data.accel[1] / 16384.0;
        float az = data.accel[2] / 16384.0;

        // Gyroscope: ±250°/s range (131 LSB/°/s)
        float gx = data.gyro[0] / 131.0;
        float gy = data.gyro[1] / 131.0;
        float gz = data.gyro[2] / 131.0;
```

```
        // Temperature: (in °C)
        float temp = data.temp / 340.0 + 36.53;

        // Compute roll and pitch using accelerometer data
        float roll = atan2(ay, az) * 180.0 / PI;
        float pitch = atan2(-ax, sqrt(ay * ay + az * az)) * 180.0 /
PI;

       /* no need to see this

        // Print results with fixed-width formatting
        print("%c", HOME);  // Clear terminal
        print("MPU-60X0 IMU Data\n");
        print("-----------------\n");
        print("Accel: X=%7.2fg  Y=%7.2fg  Z=%7.2fg\n", ax, ay, az);
        print("Gyro:  X=%7.2f°/s Y=%7.2f°/s Z=%7.2f°/s\n", gx, gy,
gz);
        //print("Temp:  %7.1f°C\n", temp);
        pause(200);
      */

      //read_imu_data(roll, pitch);
        print("Roll: %.2f°, Pitch: %.2f°\n",roll, pitch);
        pause(500);
}
```

Main RPS-GANTRY CODE

```
i) #include "simpletools.h"                      // Include
simpletools
#include "adcDCpropab.h"
#include "imu_sensor.h"


#define RPS_LEG1_PIN 12
#define RPS_LEG2_PIN 13
#define RPS_LEG3_PIN 14
#define GANTRY_PIN1 15
#define GANTRY_PIN2 16

float total_duration = 17900; //~ milliseconds (17.9 seconds)
float total_distance = 0.095; //meters
```

```
float full_speed = 1700; //pulses per millisecond (1.7 pulses per
second)

const float upThresh = 3.0;
const float downThresh = 2.0;


static volatile int forward = 0;
static volatile int backward = 0;
static volatile int up = 0;
static volatile int down = 0;
static volatile int xR = 0;

static volatile  float lrV, udV, xrV;
// Voltage variables

//const int RPS_LEG_PINS[] = {12, 13, 14};

void sensor_cog(void *par0);
void rps_cog1(void *par1);
void rps_cog2(void *par2);
void rps_cog3(void *par3);
void gantry_cog1(void *par4); //pending
void gantry_cog2(void *par5); //not written yet
void input_cog(void *par6);

//sensors
int16_t gyro_offsets[3];

//motors
//static volatile int leg1cog, leg2cog, leg3cog;
unsigned int servo1_stack_size[256];                // Stack vars
unsigned int servo2_stack_size[256];                // Stack vars
unsigned int servo3_stack_size[256];                // Stack vars
unsigned int gantry1_stack_size[256];                // Stack vars
unsigned int input_cog_stack[256];                // Stack vars

int main()                                          // main function
{
   // Initialize IMU
   mpu6050_init();

   //start cogs (lol)
```

```
   cogstart(rps_cog1, NULL, servo1_stack_size,
sizeof(servo1_stack_size));
   cogstart(rps_cog2, NULL, servo2_stack_size,
sizeof(servo2_stack_size));
   cogstart(rps_cog3, NULL, servo3_stack_size,
sizeof(servo3_stack_size));

   cogstart(gantry_cog1, NULL, gantry1_stack_size,
sizeof(gantry1_stack_size));
   cogstart(input_cog, NULL, input_cog_stack,
sizeof(input_cog_stack));

   //calibration....


 while(1){
        //print current roll and pitch
          current_pos();
          //print("lrV: %.2f \n",lrV);
          //print("udV: %.2f \n",udV);
          pause(500);
          }

}

// Function that can continue on its
// own if launched into another cog.

void rps_cog1(void *par1){
     while(1) {
          while(down == 1){
    pulse_out(RPS_LEG1_PIN, 1600); //up
    pause(20);
  }
    while(up == 1){
    pulse_out(RPS_LEG1_PIN, 1400); //down
    pause(20);
  }
  while(xR == 1){
    pulse_out(RPS_LEG1_PIN, 1400); //down
    pause(20);
  }
  while(xR == -1){
```

```
    pulse_out(RPS_LEG1_PIN, 1600); //up
    pause(20);
  }
    pulse_out(RPS_LEG1_PIN, 1500); //stop
    pause(20);
             }
  }

void rps_cog2(void *par2){
     while(1) {
          while(down == 1){
    pulse_out(RPS_LEG2_PIN, 1600); //up
    pause(20);
  }
    while(up == 1){
    pulse_out(RPS_LEG2_PIN, 1400); //down
    pause(20);
  }
  while(xR == 1){
    pulse_out(RPS_LEG2_PIN, 1600); //up
    pause(20);
  }
  while(xR == -1){
    pulse_out(RPS_LEG2_PIN, 1400); //down
    pause(20);
  }
    pulse_out(RPS_LEG2_PIN, 1500); //stop
    pause(20);
             }
  }

void rps_cog3(void *par3){
     while(1) {
          while(down == 1){
    pulse_out(RPS_LEG3_PIN, 1600); //up
    pause(20);
  }
    while(up == 1){
    pulse_out(RPS_LEG3_PIN, 1400); //down
    pause(20);
  }
  while(xR == 1){
    pulse_out(RPS_LEG3_PIN, 1600); //up
```

```
    pause(20);
  }
  while(xR == -1){
    pulse_out(RPS_LEG3_PIN, 1400); //down
    pause(20);
  }
    pulse_out(RPS_LEG3_PIN, 1500); //stop
    pause(20);
             }
  }

void gantry_cog1(void *par4){
  while(1){
    while(forward == 1){
    pulse_out(GANTRY_PIN1, 1600); //RIGHT
    pause(20);
  }
    while(backward == 1){
    pulse_out(GANTRY_PIN1, 1400); //left
    pause(20);
  }
    pulse_out(GANTRY_PIN1, 1475); //stop
    pause(20);
}
}

void input_cog(void *par6){
  pause(1000);                                    // Wait 1 s for
Terminal app
  adc_init(21, 20, 19, 18);                       // CS=21,
SCL=20, DO=19, DI=18

    while(1)                                        // Loop
repeats indefinitely
    {
      udV = adc_volts(1);                           // Check A/D 0
      lrV = adc_volts(0);                           // Check A/D 1
      xrV = adc_volts(2);                            //Check A/D 2
      if(lrV > upThresh){
        forward = 1;
        backward = 0;
      } else if(lrV < downThresh){
        backward = 1;
```

```
      forward = 0;
    }  else {
      forward = 0;
      backward = 0;
    }

    if(udV > upThresh){
      up = 1;
      down = 0;
    } else if(udV < downThresh){
      down = 1;
      up = 0;
    }  else {
      up = 0;
      down = 0;
    }

    if(xrV > upThresh){
      xR = 1;
    } else if(xrV < downThresh){
      xR = -1;
    }  else {
      xR = 0;
    }

  }
}
```