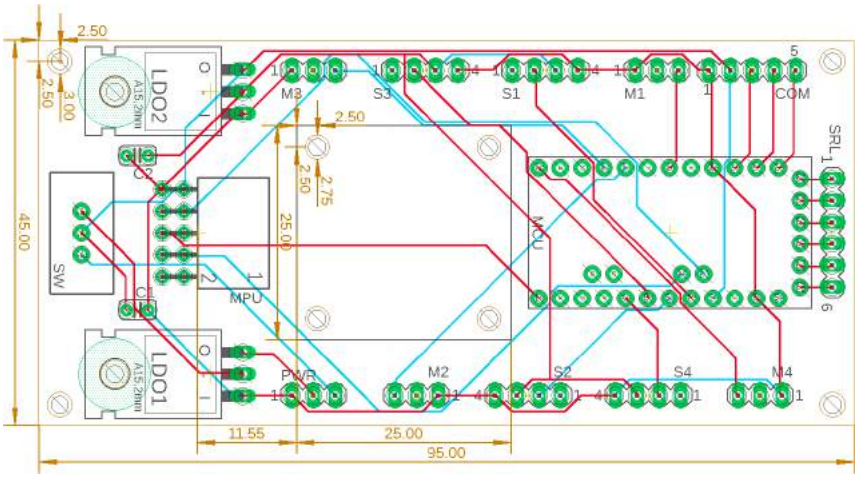
Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project Report 2019



Project Title:	**Electronics for a Robotic Quadrotor-Top Inverted Pendulum (QTIP)**
Student:	**Jee Yong Park**
CID:	**00943476**
Course:	**EE4**
Project Supervisor:	**Dr. Eric C. Kerrigan**
Second Marker:	**Dr. Adrià Junyent-Ferré**

Acknowledgements

I would like to thank Dr. Eric Kerrigan and Ian McInerney for the insight and guidance they have provided throughout the length of this project.

I would also like to thank Charlie Bell for the preceding work and its documentations upon which this project is based.

For the last but not the least, it should be acknowledged that this would not have been possible without the patient and loving support from my family, and from my dearest friend and love, Pearamon 'Sasha' Tulavardhana.

Abstract

This project aims to make improvements and adjustments to an existing quadrotor top inverted pendulum self-balancing robot [1], and utilise the said hardware platform to implement an effective linear feedback controller.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Problem overivew

The quadrotor-top inverted pendulum robot (QTIP) has been developed in [1]. As illustrated in Figure 1, it consists of a 3-DOF (roll, pitch, and yaw) top inverted pendulum mounted via a ball joint on a 3-DOF (translation in 2 axes and rotation) cart. The pendulum is actuated by four rotors to balance itself in a similar arrangement to that of most quadrotor UAV's, but with the direction of the rotors outwards and orthogonal to the pendulum arm, and the cart is actuated by two wheels driven by separate DC motors. Further details of the original system can be found in Section 2.2.

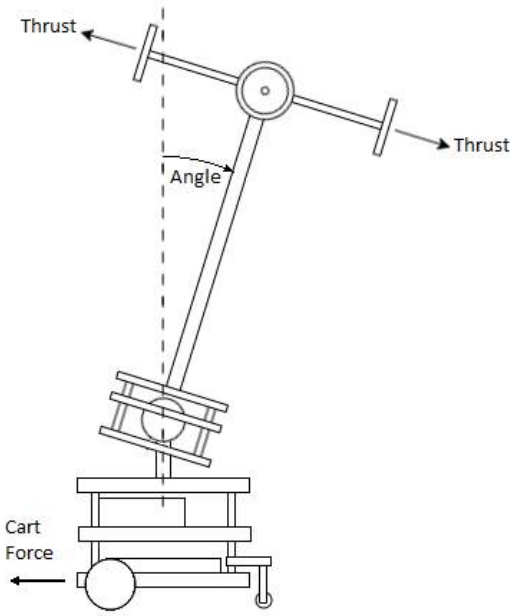


Figure 1: Hardware concept design of the QTIP system [1]

1.2 Overview of project goals

The aim of this project are defined as below:

1. Devise and implement hardware improvements upon Bell's QTIP.
   (a) Separate the pendulum and cart systems to be run by different microcontrollers.
   (b) Devise and implement mechanical and electrical safety measures.
   (c) Improve the robustness, modularity, and repairability of the electronics.
   (d) Optimise the software to the improved hardware.
2. Identify and derive the system model of QTIP.
3. Design and implement effective linear control algorithm.

The QTIP system has several design issues that hinders its safety, modularity, and robustness. These are to be addressed such that QTIP can be used as a reliable platform on which different control algorithms can be tested. Software changes and optimisations must follow the hardware

updates, as well as refining the less than ideal state of the existing source code. The specific objectives as to achieving such are defined in detail in Section 2.2.

When the refinements are made to QTIP, it needs to be confirmed whether or not it is a suitable platform on which controllers can be tested. This will be put to test by attempting to design and implement an effective stabilising linear feedback controller.

In order to design suitable controller for the system, the system model needs to be identified - including the system parameters and the model dynamics - and linearised.

1.3 Report structure

This report begins in Section with the review of relevant literature in terms of analysing and controlling the system, the overivew of Bell's QTIP and elaborating on the required improvements and what is needed to implement them.

It will then go into further detail as to what improvements or adjustments were made to the robot and how in Section 3, for both the hardware and the software.

Having discussed the hard and soft implementations made to the QTIP, Sections 4 goes on to discuss the system model derivations and the control design, detailing on how the model parameters are obtained and controller weights are tuned.

Section 5 elaborates how the control measures designed in the previous section be tested in a set of step-by-step instructions, for which the results are laid out in Section 6 and its implications are discussed in Section 7.

The report then concludes with a qualitative overview of this project in 8.

2 Background

2.1 Relevant previous works

While the project is based on the previous developments made in [1], an attempt in building a very similar system has been made in [2], which contains great details of the system dynamics derivation specific to QTIP. [3] also provides good insight into modelling of drone rotor thrust. As the system uses Euler angles to describe its positional states, some knowledge in rotation matrices based on Euler angles is required to model the system dynamics as well, which can be obtained from [4].

In this report, the model derivation is carried out based on Euler-Lagrange equation of motion, for which [5] provides good introduction. As for the control design, [6] goes into great depth as to how PID controls can be implemented and tuned.

2.2 Overview of Bell's QTIP

2.2.1 Hardware

Figure 2: Overall view of Bell's QTIP robot

The QTIP is composed of largely two hardware systems, the cart and the pendulum, as described in Section 1.1. The cart carries a fused power switch unit, a lithium-polymer battery, a power distribution board (PDB), a pair of DC motors and drivers, a Bluetooth communications module, and a microcontroller unit (MCU) board. The weight of the pendulum is minimised by placing only the motion processor unit (MPU) board, the rotors, and the ESCs on the pendulum. Communications and control of all components are managed by the MCU (Arduino Mega 2560 Rev3). Details of the mentioned components can be found in Table 1 of [1].

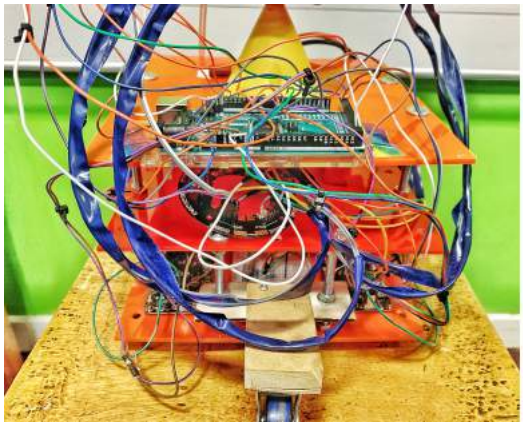
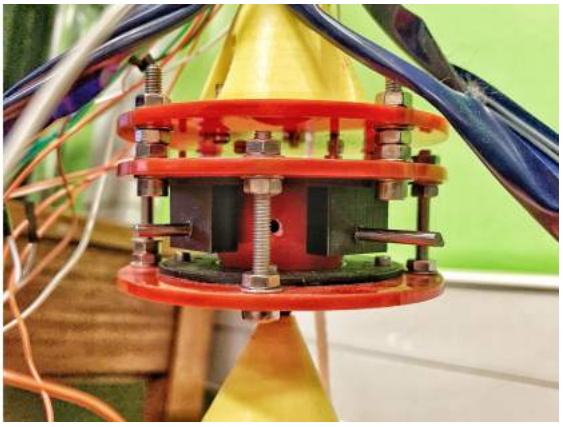



Figure 3: Cart system and Arduino Mega 2560 Rev. 3 microcontroller board (*left*), ball joint for the pendulum (*right*)

2.2.2 Hardware improvements

The largest outstanding problem of the original robot is the large number of wires and components loosely hanging around and along the pendulum, as can be seen in Figures 2 and 3. In addition to the ESCs being loosely taped to the pendulum, there are 8 high-current wires powering the rotors, 8 wires providing PWM signals to the ESCs from the microcontroller board, and 5 wires connecting the digital signals between the motion processor board (MPU6050-BB) and the microcontroller board. This not only can amount to electrical safety issues, but also makes the system very difficult for modifications or repairs, and the long, loose transmissions can be prone to electromagnetic noises from the high-powered motors.

There are several propositions to address the said problem. First of which is to install a new MCU on the pendulum to drive the rotors, as opposed to the Arduino Mega 2560 Rev3 board driving both the pendulum and the cart. This will keep the signal transmission lines between the ESCs, the MCU, and the MPU only on the top of the pendulum instead of along it.

Second is to install a new power distribution board on the pendulum. This can minimise the length and number of power transmission lines from the battery on the cart and to the components on the pendulum.

Third is to design, fabricate, and assemble a printed circuit board (PCB) on which the new MCU and the MPU will be mounted. This will eliminate loose wires between the MPU, MCU, and the ESCs, minimise the number of external connections to power source, and enable for more modular and robust connections between other components via header pins and sockets.

2.2.3 Software

QTIP, being based on an Arduino microprocessor, runs on source codes written in C++ based on Arduino libraries. The latest version of the source code lets the cart be driven with an Android mobile application via the HC-05 Bluetooth module, while taking in commands from PC via USB serial connection to start the PID controller for the ESCs.

It should be noted that the source code utilises the MPU6050 and Arduino I2Cdev libraries from the I2C Device Library [7] for the I2C communication between the MPU and the Arduino Mega. The library enables the Arduino Mega to initialise the connection with the motion processor board and access Euler angle information computed by the on-board digital motion processor (DMP).

The PID controller algorithm sets a positive offset and minimum and maximum saturation for each rotors, and applies feedback according to the PID gains set either by default or set by the user. The difference of the thrust of two opposing rotors in the direction of either the roll or the

pitch becomes the two system inputs. The state feedback takes the Euler angle state readings directly from the DMP and subtracts the reference values to obtain error signal, integral feedback is computed by summing the error signal multiplied by the time taken for each loop iteration, and derivative feedback divides the difference of the state since the last iteration by the last loop duration. It does stabilise the pendulum as well as reject external disturbance to a certain degree, though there is some unwanted offset to the equilibrium.

2.2.4 Software improvements

Bell's software source code that drives the QTIP is able to drive the cart and stabilise the pendulum, but has several shortcomings. The most important of which is that it was written only for the Arduino Mega board. As a new MCU will be processing the controller algorithm for the pendulum system, the source code needs to be adjusted and optimised to the new MCU such that the duration of each iteration is short enough and consistent, and the software does not take up too much of the MCU's memory. A new software also needs to be written for the Arduino Mega to drive the cart and enable communication with the new MCU, either for more feedback information for the controller or for user command inputs that need to be sent to the new MCU.

Bell's software also has shortcomings in the code itself that need to be fixed. There are many redundant, unused, or unnecessary parameters and lines that either take up the MCU's memory or slow down the performance. Lack of functional modularity and adherence to general coding conventions also reduces the source code's readability and can be an obstacle to the developer modifying the software.

It is also likely that any newly installed controller software can fail and force the robot to lose stability while the rotors are active, and possibly cause damage to either human or hardware. Thus a software-based remote kill-switch is to be devised and implemented such that the rotors can be deactivated by the user any time.

2.3 Hardware components survey

Arduino Pro Mini 328 5V/16MHz board, based on the ATMega328 microcontroller, was deemed fit for the specification. It is very small and light and hence does not add too much weight to the pendulum, has both serial and I2C pins which respectively are to be used for communications with Arduino Mega and MPU6050, has enough number of PWM I/O and digital pins required to drive the motors and Sharp GP2Y0A60SZ IR sensors (as proposed in [1]), shares the same supply voltage and clock rate with the Arduino Mega at 5V, and has enough computational capability to run the controller software.

HobbyKing universal 12-way 120A multirotor power distribution hub, which is already used to power the original QTIP robot, is an adequate candidate for the new PDB to be installed on the pendulum. It has a universal battery elimination circuit (UBEC) that supplies 5V DC that can power both the MCU and the MPU, has two pairs of main power supply connections such that the PDB on the cart can maintain connection to both the other PDB and the battery supply, and has enough high-current connections for the ESCs which has already been proven to work reliably in [1]. The board is also in a shape of a ring, which makes it especially easy to mount on the pendulum, with the pendulum arm running through it.

Components that were chosen for the PCB assembly are listed on Table 1. 1μF ceramic capacitors should provide the capacitative DC coupling to reduce noise induced by the motors and the ESCs, and two low-dropout voltage regulators are placed to power the IR sensors that run on 3V DC supply, and in case the user needs to power the board only from a direct power supply from the battery. The single-pole double-throw (SPDT) switch of adequate current ratings is placed such that the user can switch the board's power supply from that of the PDB or the aforementioned external direct supply.

2.4 Software tools

Mathworks MATLAB Live Script is an ideal tool to facilitate the model derivations and controller designs, which provides powerful parameter types and functions with MATLAB Symbolic Math and Robust Control Toolboxes.

Arduino IDE is an essential software for compiling, programming, and monitoring Arduino devices. It comes with default compiler and programmer settings for different Arduino devices, and has a built-in serial monitor for the USB serial communication.

PuTTY is an SSH and telnet client that is very useful as it can monitor serial connection via USB and automatically save the communication log, for instance to plot real-time data that the MCU prints to the USB serial connection, unlike Arduino IDE's built-in serial monitor.

Autodesk EAGLE is a computer aided drawing (CAD) software developed specifically for PCB development. A vast range of user-defined libraries of component symbols, footprints, and values is available to import to project, and design files necessary for manufacturers to fabricate the designed PCB can directly be exported.

3 Improvements and Adjustments

3.1 Hardware improvements

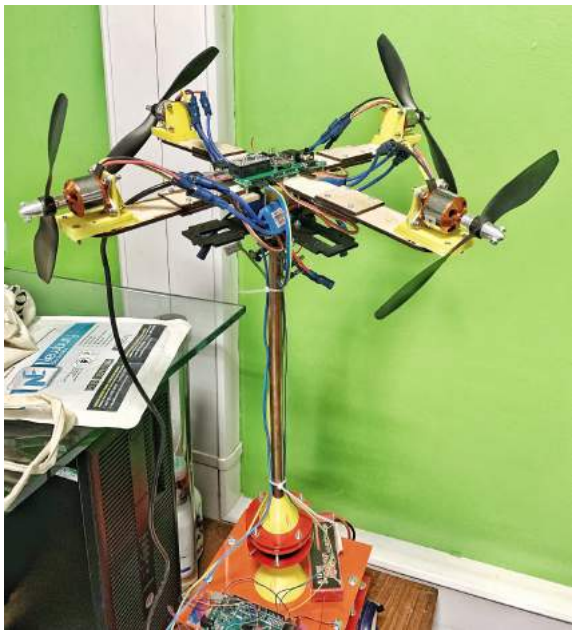


Figure 4: Overall look at the changed hardware

3.1.1 Cart and pendulum separation

Arduino Pro Mini 328 was mounted on top of the pendulum to control the pendulum actuation separately from that of the cart system. Figure 5 shows the PCB on which the MCU is mounted as detailed in Section 3.1.3. Pins 3, 5, 6, and 9 give 8-bit PWM outputs and are connected to ESCs that are driven by them. Pins A4 and A5 give SDA and SCA connections necessary for I2C communication with the MPU, and pin 10, set to digital output, gives the digital interrupt signal that [what does it do?]. TX0 and RX0 pins are used for serial connection with the Arduino Mega, and RST pin provides the reset control that can be used by Arduino Mega to enable and disable the Pro Mini when required. Pins A0, A1, A2, and A3 are capable of analogue-to-digital conversion (ADC), which is to be connected to the IR sensors proposed in [1], although they are not used in this project. Section 3.1.3 further details how the pins are connected to the necessary components.

It should be noted, however, that the Arduino Pro Mini only has one pair of serial connections (TX0 and RX0), which are occupied should it connect to PC via USB, and USB serial connection will not work if the TX0 and RX0 pins are connected elsewhere. Hence the FTDI breakout board should be disconnected when the serial is occupied for communication with Arduino Mega and vice versa.

The adjustment proves to provide much more robust connection between the MCU, the MPU, and the ESCs, and rids of the need of 15 wires that ran along the pendulum.

3.1.2 Power supply

As can be seen in Figure 6, the original PDB has been mounted on the pendulum, and an additional, identical component has replaced the original. The original proposition was only to move the original PDB to the pendulum, but that required additional power connections from the UBECs of

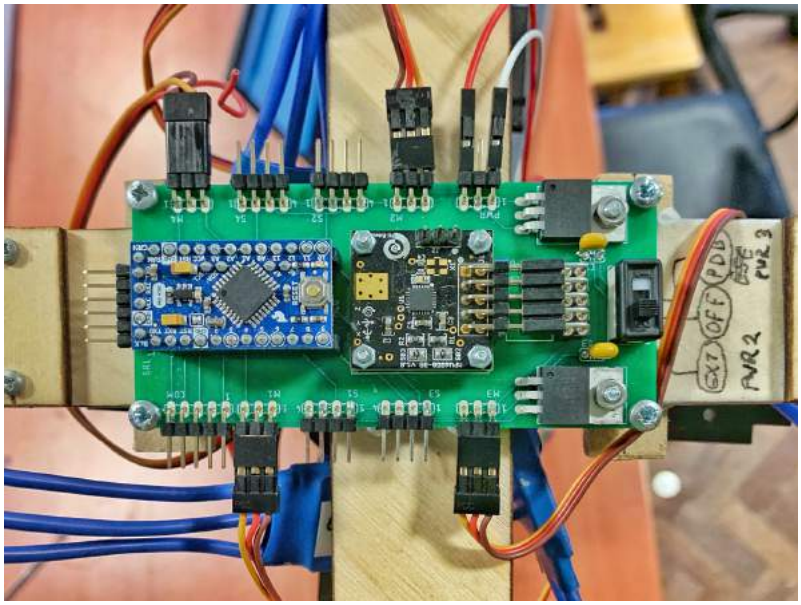


Figure 5: Arduino Pro Mini 328 microcontroller unit and MPU6050-BB motion processor unit mounted on the MCU mount PCB

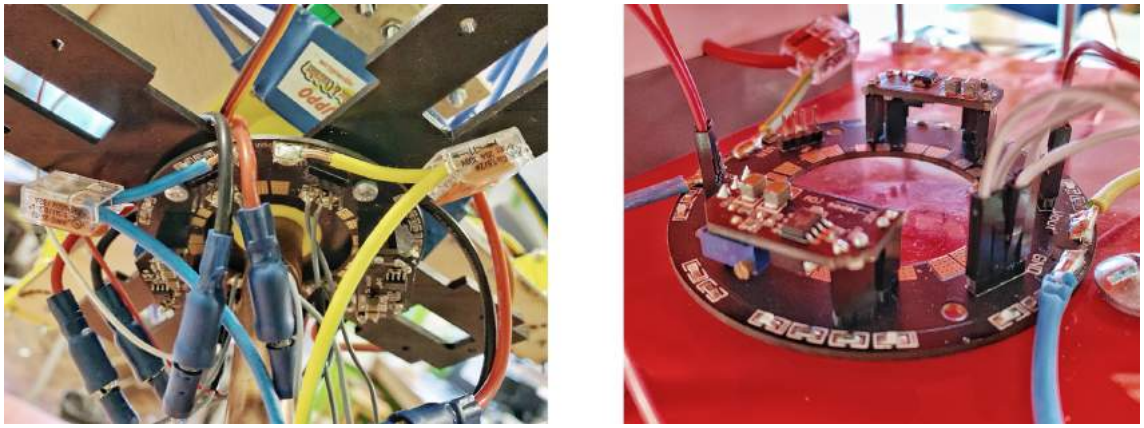


Figure 6: Original PDB mounted on the pendulum (*left*), and new replacement PDB connected to main power switch on the cart (*right*)

the PDBs to run down along the pendulum arm. The bullet connectors that linked the main power transmissions and the soldered connections between the board and the switch box were replaced with WAGO 221 terminal block connectors, which makes it easier for disassembley, should a need for adjustment arise.

3.1.3 Dedicated MCU and MPU mount PCB

Initially, the Arduino Pro Mini was mounted on the pendulum and maintained connections with the necessary components via jumper cables. This, however, was still prone to external EM noise and a large number of jumper cables still made repairs or modifications difficult. The design and implementation of PCB was proposed such that it mounts the MCU and MPU and connects to external components (FTDI breakout board, ESCs, IR sensors, power source, and Arduino Mega) via header pins and sockets. Additionally, as can be seen from Table 1, header sockets are soldered where the MPU and the MCU are positioned on the PCB, allowing for further removals of each parts should a repair or further change be required. The designs for which are detailed in the the schematics in Figures 7 and 8, and the list of components in Table 1.

A switch of appropriate power rating and low-dropout voltage regulators (LDOs) were added so that the board can be powered off or switched between the power supply from either the PDB or an external voltage supply, for instance, a direct wire from the battery without switching on the power so as to debug the MCU without activating the rotors. A 3.3V LDO was also placed, with which the IR sensors will be powered in the future. Both 5V and 3.3V supplies are decoupled with 1μF ceramic capacitors as well, which will further reduce the electronics' exposure to EM noise.

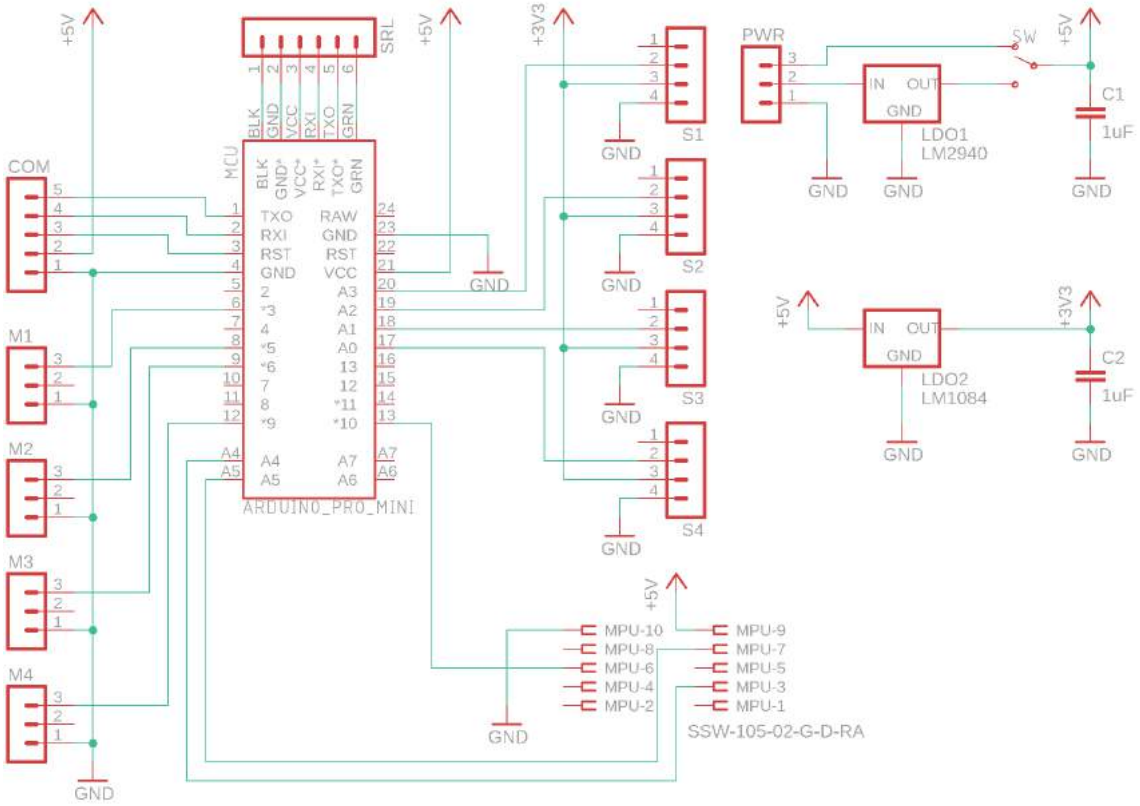


Figure 7: Circuit schematics for the MCU mount PCB

Designator	Component description	Value
MCU	Arduino Pro Mini 328P 5V/16MHz Samtec SSQ-1XX-01-S-S header socket	N/A 2, 6, 12 pins
MPU	Phi Robotics MPU6050-BB Preci-Dip 410 series header socket Preci-Dip 803 series header socket	N/A 2 rows 10 ways 2 rows 10 ways
LDO1	Texas Instruments LM2940CT-5.0/NOPB	5V output
LDO2	Texas Instruments LM1086IT-3.3/NOPB	3.3V output
C1, C2	KEMET multilayer ceramic capacitor	1μF $\pm$20%
SW	APEM 25139NAH SPDT slide switch	4A, 125VAC
SRL	Samtec TSW-1XX-L-S-RA header pin	1 row 6 ways
S1, S2, S3, S4	Samtec TSW-1XX-L-S-RA header pin	1 row 4 ways
PWR, COM, M1, M2, M3, M4	Samtec TSW-1XX-L-S-RA header pin	1 row 3 ways

Table 1: Components included in the PCB design

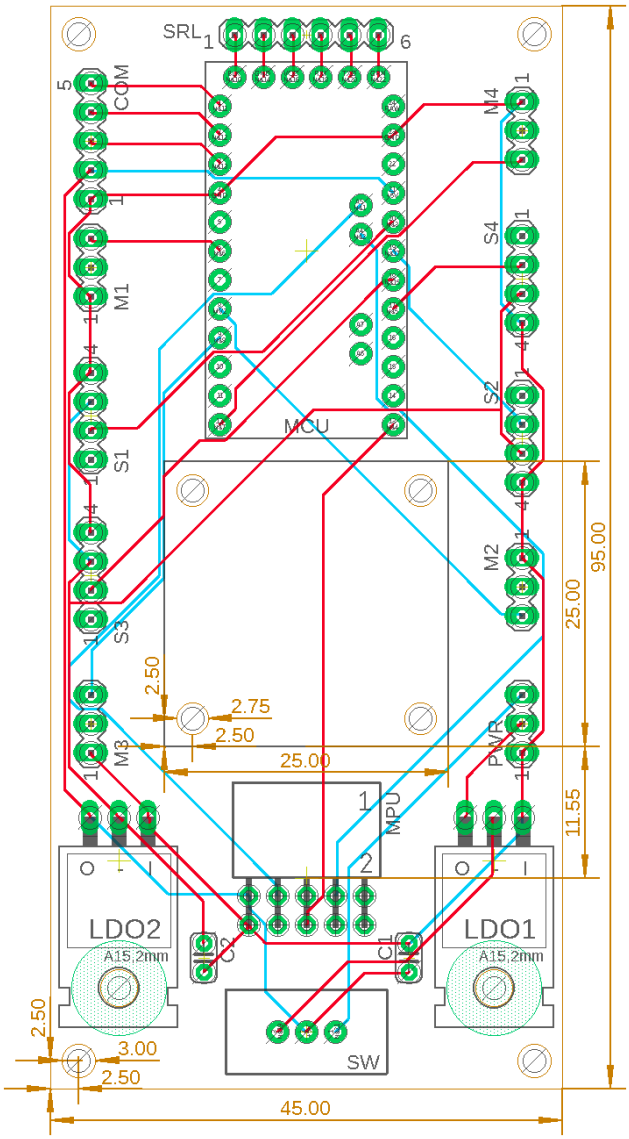


Figure 8: Board design for the MCU mount circuit board

3.2 Software improvements

3.2.1 Software optimisation for Arduino Pro Mini 328

As ATmega328P microcontroller, which powers the Arduino Pro Mini, has considerably small memory capacity compared to the Arduino Mega, it is vital that there is no wasted storage. Hence the source code had to be reviewed to remove unused or redundant variables and lines of code.

There also were worries of slowed computation on the Arduino Pro Mini, as well as problems that rise from irregular sampling periods. The original source code incorporates a timer that records the length of the previous iteration in order to compute derivative and integral feedbacks for the PID controller. Using this, experiments were thus conducted to measure the typical and maximum time taken for the MCU for one iteration, by placing timers that log the duration of each loop and saving to an array. Results, of course, varied as the structure and additional functions of the source code varied, though it could be concluded that one loop iteration is very unlikely to exceed 20ms. The timer was then used to keep the software from repeating the loop before the timer has reached the typical maximum sampling period measured for the specific source code. The example below illustrates how the period constraint is applied.

```
#define TS 20 // sampling period in milliseconds
unsigned long now = 0; // timestamp for current iteration
unsigned long then = 0; // timestamp for previous iteration
unsigned short anomalyCounter = 0; // counts how many times TS has been exceeded

void setup(){
    then = millis();
}

void loop(){
    now = millis(); // update timestamp for current iteration
    if(now - then > TS){
        anomalyCounter++;
    }
    while(now - then <= TS){
        now = millis();
    } // wait until (now-then > TS)
    mainFunction(); // pose estimation and feedback loop
    then = now; // update timestamp for previous iteration
}
```

3.2.2 Serial communications

The original source code uses the MCU's USB serial channel to receive commands from PC. It is, however, more difficult to do so with USB connection on the top of the pendulum. Hence Arduino Mega, which has 4 serial channels (including channel for USB connection), will receive user command inputs from PC over USB serial, and communicate with the Arduino Mini via the second serial channel with single-byte flag to replace these commands and system messages. Though the interface with the user is currently done with a serial communications monitor on a PC, it should be relatively simlple to modify the current source code to receive user commands via Bluetooth as the HC-05 module communicates with Arduino Mega via serial as well, and hence completely untether the system.

Additionally, having the behaviours dictated by different flags that each MCUs receive, function calls were rearranged such that the main loop decides which functions to call by putting an integer `sysStatus` into a `switch` statement, and the change in `sysStatus` is dictated by the incoming serial communications.

As an additional communications channel, a digital pin (30) on Arduino Mega was wired to the reset pin (RST) on Arduino Pro Mini. This is used by the Arduino Mega to initialise Arduino Pro Mini at wanted times by setting the digital output to low, and also as a emergency kill switch that force restarts the Pro Mini either when the user gives the designated input or when an unexpected error occurs.

The prime drawback of replacing the Mega with the Pro Mini as the pendulum's controller is that its memory is too small to store real-time data (angular states in `float`, for example) for the entire duration for which the feedback loop is active. Since serial is a byte-by-byte communication, the source code for Arduino Pro Mini casts the `float` variables to four individual bytes and writes to the serial channel using a `union` struct, as shown in the example below. When the Arduino Mega uses the same struct to convert the four bytes back to `float` variable, it has enough internal storage to save the variables in an array.

Below example code shows how the `float` type variables are sent from Arduino Pro Mini.

```
// sending from Arduino Pro Mini to Arduino Mega
float roll, pitch;
union ftb {
```

```
    float fval;
    byte bval[4];
};
ftb floatToBytes;

// cast float variable to 4 bytes and send to Serial
void sendFloat(float num){
    floatToBytes.fval = num;
    Serial.write(floatToBytes.bval[0]);
    Serial.write(floatToBytes.bval[1]);
    Serial.write(floatToBytes.bval[2]);
    Serial.write(floatToBytes.bval[3]);
}

// dump angle data to Arduino Mega (APM doesn't have enough storage)
void dumpData(){
    Serial.write(20); // flag for data dumping
    sendFloat(roll);
    sendFloat(pitch);
}
```

Below example code shows how the float type variables are received and saved by Arduino Mega.

```
// Arduino Mega receiving
#define ARRAYSIZE 600
int savedData = 0;
float angleLog[ARRAYSIZE][2]
ftb floatToBytes; // identical union struct type as for Arduino Pro Mini

// cast incoming 4 bytes in Serial1 read buffer to float
void receiveFloat(){
    while(Serial1.available() < 4); // wait until 4+ bytes have arrived
    Serial1.readBytes(floatToBytes.bval, 4);
}

// saves angle data (roll, pitch) one instance at a time
void saveData(){
    if(savedData < ARRAYSIZE){
        receiveFloat(); // receive roll
        angleLog[savedData][0] = floatToBytes.fval;
        receiveFloat(); // receive float
        angleLog[savedData][1] = floatToBytes.fval;

        savedData ++;
    }
}
```

4 Modelling of System Dynamics and Control Design

4.1 State space model derivation

The system is consisted of a cart and a pendulum, but in this project, the pendulum will be considered as the sole inertial frame of reference and regard the cart's movement as an external exertion of acceleration. For this reason the yaw of the pendulum will be disregarded.

Figure 9 below describes the frame of reference that will be used in this section to derive the state space model, in which the origin of the Cartesian space is placed at the base of the pendulum as the pendulum pitches around the x axis and rolls around the y axis, and the target equilibrium is placed on the positive side of the z axis.

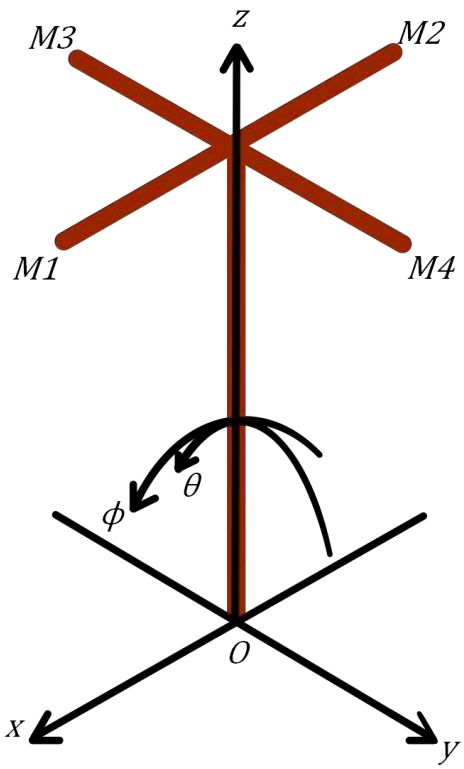


Figure 9: Frame of reference used for the model derivation, where M1, M2, M3, M4 are the rotors.

4.1.1 Assumptions

It should be noted that in order to obtain the linear state space model, it is assumed that

1. Mechanical friction that occurs at the ball joint is negligible.
2. Thrust of each rotor is proportional to the PWM signal given to the motor.
3. Drag introduced by rotors in orthogonal axis is negligible.

Assumptions 2 and 3 will be justified in the following section.

4.1.2 Parameters and variables

Parameters	Symbol	Value	Unit
Mass of pendulum	m	1.066	kg
Position of centre of gravity	r	0.35	m
Thrust constant	K	0.0074	N
Gravitational constant	g	9.81	m/s^2

Table 2: List of parameter values

Table 2 lists the system parameters needed to derive the model. Note in the table the thrust constant K. This is a constant obtained experimentally based on assumptions 2 and 3 from Section 4.1.1, and describes the proportional characteristics of individual thrust of each rotor with the Arduino Servo command used to control the ESC. The more precise characterisation of rotor dynamics is shown as equations (1), (2), and (3), which each model the motor speed, individual rotor thrust, and total thrust acted on the end of the pendulum at x-axis. [2] [3]

$$\omega_i = \frac{-\frac{1}{R_m K_V K_Q} + \sqrt{\frac{1}{R_m^2 K_V^2 K_Q^2} - 4K_d(\frac{i_f}{K_Q} - \frac{V_{quad} u_i}{u_{max} R_m K_V})}}{2K_d} \tag{1}$$

$$f_i = K_T \omega_i^2 + \delta_T v_i \omega_i \tag{2}$$

$$f_x = K_T(\omega_2^2 - \omega_1^2) + \delta_T(lq - \dot{x}_m)(\omega_2 - \omega_1) + \frac{K_d}{l}(\omega_4^2 - \omega_3^2) \tag{3}$$

where R_m, K_V, K_Q, V_{quad}, δ_T, K_T, K_d, v_i, x_m, ω_i, f_i, and f_x each are [symbol meanings] respectively.

The pendulum being a 2 degrees of freedom system with roll and pitch, the state x will be expressed as

$$\mathbf{x} = \begin{pmatrix} \phi \\ \dot{\phi} \\ \theta \\ \dot{\theta} \end{pmatrix} \tag{4}$$

4.1.3 System dynamics

The system dynamics is modelled using the Euler-Lagrange equation as characterised in Equation 5.

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = \tau_i \ , \quad i = 1, 2 \tag{5}$$

$$L = KE - PE \tag{6}$$

Note that i an interger defined in the range $[1, 2]$ as the system is considered to have 2 degrees of freedom.

First the rotation matrices are defined that transform the coordinates according to pitch and roll, and then the total rotation matrix as defined by Equation 8. [4]

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}, \ R_y(\phi) = \begin{pmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{pmatrix} \tag{7}$$

$$R = R_y R_x \tag{8}$$

Define $\mathbf{e_z}$ as the unit vector on the z-axis, as is the initial, equilibrium and target position of the pendulum. Then, when the rotations by pitch θ and roll ϕ are applied in the said order, the resulting unit vector $\mathbf{e}$ is as follows.

$$\mathbf{e} = R\mathbf{e_z} \tag{9}$$

$$= \begin{pmatrix} \cos\theta \sin\theta \\ -\sin\theta \\ \cos\phi \cos\theta \end{pmatrix} \tag{10}$$

Hence, the velocity of the pendulum in the Cartesian frame can be characterised as below.

$$\mathbf{v} = \frac{d}{dt}(r\mathbf{e}) \tag{11}$$

$$= \begin{pmatrix} r\dot{\phi}\cos\phi\cos\theta - r\dot{\theta}\sin\phi\sin\theta \\ -r\dot{\theta}\cos\theta \\ -r\dot{\phi}\cos\theta\sin\phi - r\dot{\theta}\cos\phi\sin\phi \end{pmatrix} \tag{12}$$

Kinetic energy KE can be obtained from (11), and the potential energy PE from the z componet of (9).

$$KE = \frac{1}{2}m\mathbf{v}'\mathbf{v} \tag{13}$$

$$PE = gmr\cos\phi\cos\theta \tag{14}$$

From (6), the resulting Lagrangian function is thus

$$L = \frac{mr^2\dot{\phi}^2\cos^2\theta}{2} + \frac{mr^2\dot{\theta}^2}{2} - gmr\cos\phi\cos\theta \tag{15}$$

Solving (5), $\ddot{\theta}$ and $\ddot{\phi}$ can be expressed in terms of the components of $\mathbf{x}$. Hence, the resulting system dynamics,

$$\dot{\mathbf{x}} = \frac{d}{dt}\begin{pmatrix} \phi \\ \dot{\phi} \\ \theta \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \dot{\phi} \\ \dfrac{mr^2\dot{\phi}\dot{\theta}\sin 2\theta + gmr\cos\theta\sin\phi + Ku_x}{mr^2\cos^2\theta} \\ \dot{\theta} \\ \dfrac{-\frac{mr^2\dot{\phi}^2\sin 2\theta}{2} + gm\cos\phi\sin\theta + Ku_y}{mr^2} \end{pmatrix} \tag{16}$$

Note the Ku_x and Ku_y terms in (16). This comes from the assumptions 2 and 3 made in Section 4.1.1, rationlised with the experimental results provided in Section 4.1.2. As the resultant thrust was measured at the centre of gravity along the pendulum, there is no need to factor in the length of the pendulum at which the motors are placed.

4.2 Linearisation of state space model

Following the result (16) of the previous section, the linearised state space model is obtained by evaluating the Jacobian matrix of (16) with respect to all $\mathbf{x}$ and $\mathbf{u}$ variables at the target equilibrium point, that is to say,

$$\mathbf{x} = 0, \text{ and } \mathbf{u} = \begin{pmatrix} u_x \\ u_y \end{pmatrix} = 0 \tag{17}$$

The resultant state space matrices A and B, having evaluated the said Jacobian, are as follows.

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ \frac{g}{r} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{g}{r} & 0 \end{pmatrix} \tag{18}$$

$$B = \begin{pmatrix} 0 & 0 \\ \frac{K}{mr^2} & 0 \\ 0 & 0 \\ 0 & \frac{K}{mr^2} \end{pmatrix} \tag{19}$$

It is easy to show that the C and D are identity and zero matrices of appropriate dimensions respectively.

4.3 PID control design

Though the original controller algorithm already had a tuned set of PID gains in place, it did not continue to work as effectively with the updated system, possibly because of heavier pendulum and slightly higher centre of gravity, and most likely a different sampling period. Thus the below gains are redetermined experimentally to achieve the best stabilising properties.

$$K_P = 55 \tag{20}$$
$$K_I = 0.01 \tag{21}$$
$$K_D = 8000 \tag{22}$$

4.4 LQ optimal control design

Using the derived linear state-space model, MATLAB provides in-built function `lqrd` to obtain discrete LQR gain for sampled continous-time systems [8]. The code below shows how the K_{LQR} is obtained from user-defined Q, R weight matrices and the state space matrices.

```
sys = ss(A,B,C,D);
sysd = c2d(sys,Ts);

%% LQ optimal controller
% define cost function matrices
Q = diag([250 200 250 200]);
R = 0.1*eye(2);
N = zeros(4,4);

% check ctrl'bility & obsv'bility for LQR
Co = ctrb(A,B);
unco = length(A) - rank(Co); % should be zero
Ob = obsv(A,chol(Q));
unob = length(A) - rank(Ob); % should be zero

% compute lq optimal controller (discretised)
[Klqr,S,e] = lqrd(sysd.A,sysd.B,Q,R,Ts);
[K,S2,e2] = lqr(A,B,Q,R);
```

The designed LQR, however, could not control the system to functional level and hence is omitted from Section 6.

5 Testing Procedures

5.1 Test to confirm normal operation

This section describes the test procedure used to establish that functionalities of different parts of QTIP are kept as usual after any significant software or hardware modifications are applied, as described in Section 3. Note that a special set of source codes are written specifically to test each of the aspects of QTIP that requires testing as elaborated in the following sections.

5.1.1 DC motors

`TestDC.ino` is a simple source code that writes forwards and backwards command to the DC motor drivers for a certain period of time.

1. Place QTIP on a raised platform such that the wheels are not in contact with any surface.
2. Upload `TestDC.ino` to Arduino Mega.
3. Confirm whether the wheels are turning correctly according to the serial message.

5.1.2 ESCs and rotors

`TestRotor.ino` initialises the MCU's connections to the ESCs and writes rotor outputs at low speed, one at a time for 1 seconds each.

1. Ensure the serial connection between Arduino Pro Mini and Arduino Mega is physically disabled as this will disable the USB serial connection required to upload the required software.
2. Upload `TestRotor.ino` to Arduino Pro Mini.
3. Hold onto the pendulum arm before starting the test.
4. Load the serial monitor and follow the instruction.

5.1.3 MPU connection and DMP readings

`TestMPU.ino` initialises connection to the DMP, and prints the set of Euler angle values to the USB serial in real time.

1. Ensure the serial connection between Arduino Pro Mini and Arduino Mega is physically disabled as this will disable the USB serial connection required to upload the required software.
2. Upload `TestMPU.ino` to Arduino Pro Mini.
3. Load the serial monitor and follow the instruction.
4. Move the pendulum to confirm the Euler angle readings are correct.

5.1.4 Serial connection between Arduino Mega and Pro Mini

`TestSerialPend.ino` and `TestSerialCart.ino` each initialises the respective serial channels of Arduino Pro Mini and Arduino Mega, reads the float value that user inputs via USB serial to the Arduino Mega, casts `float` into four `bytes` and sends it to the Arduino Pro Mini. The Arduino

Pro Mini then converts the received four bytes back into float, adds 1, and sends the float back to Arduino Mega, which is then printed on Mega's USB serial monitor, establishing that the serial connection between the two MCUs are correct and the float casting functions are working.

1. Ensure the serial connection between Arduino Pro Mini and Arduino Mega is physically disabled as this will disable the USB serial connection required to upload the required software.

2. Upload `TestSerialPend.ino` to Arduino Pro Mini and `TestSerialCart.ino` to Arduino Mega.

3. Disconnect Arduino Pro Mini's USB serial breakout board and reconnect serial connection channels (TX0 and RX0 on Arduino Pro Mini to RX1 and TX1 on Arduino Mega).

4. Confirm whether the received value is correct. Serial monitor will also print `true` if the received `float` is correct.

5.2 Test for controller performances

This section describes the test procedure used to quantitatively measure the performance of the stabilising linear controllers designed in Section 4. The user must ensure proper safety measures are in place before starting up the ESCs to prevent physical injuries from the rotating blades.

5.2.1 Steady-state response test

This test simply runs the controller feedback loop for 20 seconds at equilibrium to see if there exists any drift or oscillation.

1. Ensure there is no person or obstacle in the vicinity of QTIP.

2. Ensure the serial connection between Arduino Pro Mini and Arduino Mega is physically disabled as this will disable the USB serial connection required to upload the required software.

3. Ensure the power switch of the MCU mount board is in OFF position (middle).

4. Ensure all function calls to `testCart()` and `driveCart()` are commented out.

5. Upload `PID2DOFPend.ino` to Arduino Pro Mini.

6. Upload `PIDCart.ino` to Arduino Mega.

7. Disconnect Arduino Pro Mini's USB serial breakout board and reconnect serial connection channels (TX0 and RX0 on Arduino Pro Mini to RX1 and TX1 on Arduino Mega).

8. Switch the MCU mount power to PDB and switch on the main power supply.

9. Load Arduino Mega's USB serial communication with PuTTY. When setup process ends, hold the pendulum up straight and enter any character to start the feedback loop. If the setup process fails, restart the serial monitor, which will reset both MCUs.

10. If pendulum loses stability, enter 1 in the serial monitor to reset Arduino Pro Mini and shut down the ESCs.

11. Convert the saved PuTTY log to desired file name and change the extension to `.csv`.

5.2.2 Disturbance injection response test

This test gives the pendulum around 4 seconds to reach steady state and then injects external disturbance by driving the cart forwards at 23.5% duty cycle for 0.3 seconds. The duration of feedback loop is also 20 seconds.

1. Follow steps 1 through 3 of the instructions for Section 5.2.1.
2. Ensure all function calls to `testCart()` is enabled and `driveCart()` disabled.
3. Follow steps 5 through 9 of the instructions for Section 5.2.1.
4. If pendulum loses stability, enter 1 in the serial monitor to reset Arduino Pro Mini and shut down the ESCs.
5. Convert the saved PuTTY log to desired file name and change the extension to `.csv`.

6 Results

6.1 Nominal operation

- DC motors operate normally as beforehand.
- ESCs and rotors operate normally as beforehand most of the time.
  - There were instances, however, of one of the rotors not spinning properly. It is suspected that the three-phase signals are not in sync and the ESC is unable to throttle the motor, but the exact cause could not be found. This issue usually can be resolved either by swapping the ESC with that of another motor, or swapping the ESC's power connection with that of another.
- MPU connection and DMP readings are unchanged.
  - There seems to exist an offset on both axes of around 2 to 5 degrees that persisted since the previous iteration of the software and hardware.
- Serial connection between Arduino Mega and Pro Mini work normally.
  - One should be cautious of when to send a flag from the Mega as it takes longer for the Pro Mini to initialise its serial buffer since reset.

6.2 Steady-state response

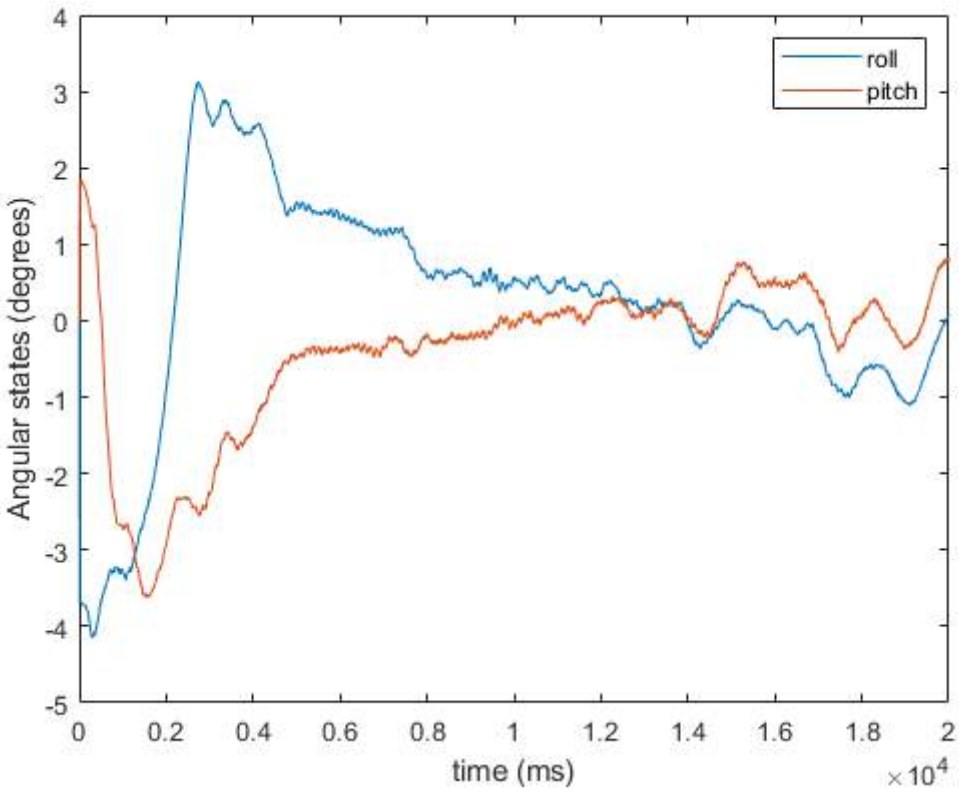


Figure 10: Steady-state response of the 2DOF pendulum

Figure 10 shows that once it reaches the steady state the roll and pitch does not exceed $\pm 1°$.

6.3 Disturbance rejection

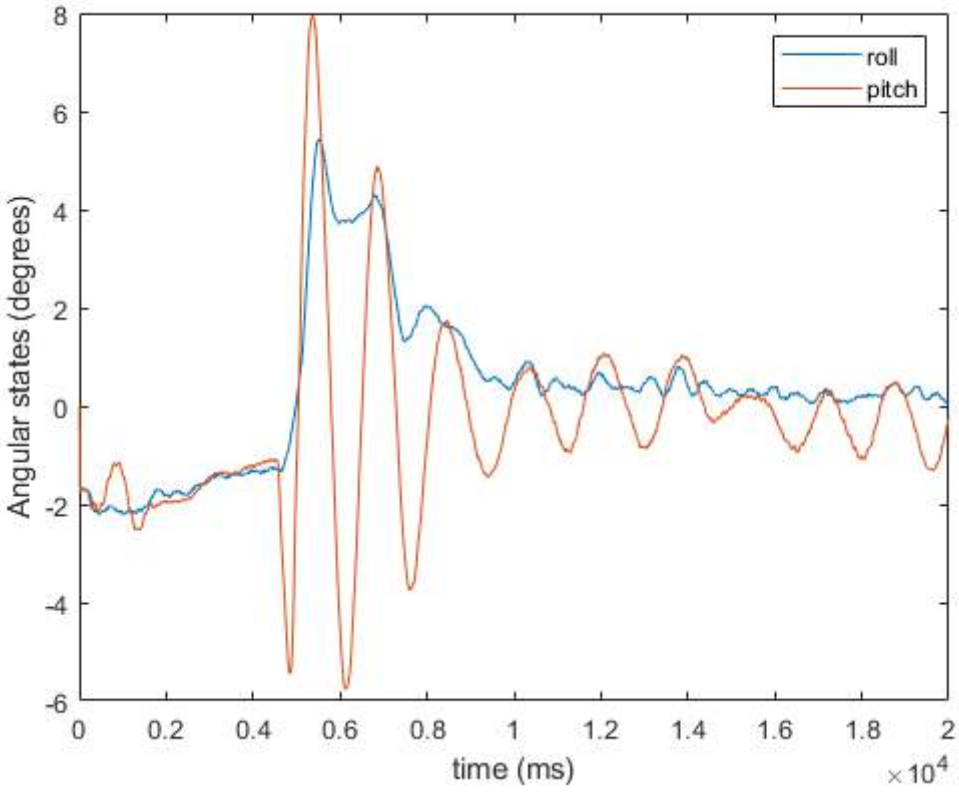


Figure 11: Disturbance injection response of the 2DOF pendulum

Figure 11 shows steady state at around $-1.5°$ until the jolt induced by the sudden acceleration and deceleration of the cart, where it can be seen that the period of oscillation is twice that of the cart activation time. Fast recovery of reference tracking can be ovserved on the roll and pitch also sees reduction from peak-to-peak $14°$ to $2°$ within 6 seconds.

7 Evaluation

Although the limitation of the MCU's internal storage led to the limitation of time window for which the responses could be observed, the results show that the designated PID gains can in fact stabilise the pendulum in the exposure to external disturbance to a certain degree.

7.1 Limitations

There are several variables that could have factored into the undesired behaviours of the controller. First of which is having made too many assumptions when modelling the system. All frictions and drags were ignored in the derivation of the system dynamics, which in real life does have observable impacts.

Other reasons may include the shift of centre of gravity as the cables move, or in fact the tension from the cables exerting force on the pendulum, that the thrust varies as battery drains, the high and low saturation points for rotor inputs that are not accounted for in the model, and the offset in the DMP readings.

It should be noted that system does become unstable if the cart's disturbance injection is increased to 0.5 seconds from 0.3, and the pendulum would often rotate around z axis while stabilising itself.

8 Conclusions

8.1 Summary of project

The QTIP has recieved great amount of modifications through this project. Much of the wires and components have been decluttered, modularised, and made easy to work with in general. In fact, not only the hardware, but also the software, has become much easier to manipulate and modify. As a platform for students to test control designs on, that should most definitely be the right direction of change for what was left from the previous iteration.

The greatest challenge that I did face over the course of the project would have been that the robot would behave in a completely different way than what I think it would most of the time, and that it can usually be resolved with a method that is just as unexplainable. This, though, was a great lesson in terms of building and engineering a complex system in that some uncertainty is bound to happen.

It does, however, come with quite a few regrets. Mostly so in that much of the work was iterative refinement based on the previous work, and nothing completely new was added to it, in which sense it is indeed regrettable that although a discrete linear quadratic regulator was in fact designed, it could not be tuned to a functional level.

8.2 Future work

Even though the project largely centred around improving QTIP from what was created in [1], there still are some hard and soft improvements left to be done. Hardware-wise, the robot can benefit from some more safety measures. These include casings for the rotors to prevent physical injury, suspension for the system to prevent hardware damage should it topple over, installing a relay switch circuit to completely cut off power supply for emergency, and insulating the exposed high-current connections and the pendulum arm for electrical safety. The casing and suspension in particular, could benefit from a wire fence design such as [9].

Even though the hardware basis for integration of IR distance sensors have already been implemented in [1] as well as the new sensor and microcontroller mount board, this is yet to be embedded into the pose estimation algorithm of the pendulum. The system has not benefitted from the HC-05 Bluetooth module in this project either.

This project focused on re-tuning an existing linear controller for the pendulum as a 2DOF system. The other 4DOF can be incorporated into the model such that when the user moves the cart to a certain position, the cart also controls the speed of the DC motors so as to keep the pendulum stable. Other modes of control can also be explored, such as $\mathrm{H}_i nfty$ optimal control or linear quadratic Gaussian control, for which the different performance measures could be compared.

References

[1] C. Bell, "Construction and control of a 2DOF inverted pendulum with propellers and a ground robot," M.Sc Project Report, Department of Electrical and Electronic Engineering, Imperial College London, 2018.

[2] P. F. Uhing, "Design, modeling, and control of a two degree of freedom pendulum on an omnidirectional robot," M.Sc Thesis, Department of Electrical and Computer Engineering, owa State University, 2016.

[3] M. Rich, "Model development, system identification, and control of a quadrotor helicopter," M.Sc Thesis, Department of Electrical and Computer Engineering, Iowa State University, 2012.

[4] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[5] A. Brizard, *An Introduction to Lagrangian Mechanics*. World Scientific, 2008. [Online]. Available: https://books.google.co.uk/books?id=JfQkh-qY8ucC

[6] K. J. Åström and R. M. Murray, *Feedback Systems*, 2nd ed. Princeton, NJ [u.a.]: Princeton Univ. Press, 2008.

[7] J. Rowberg, "I2C device library collection for AVR/Arduino or other C -based MCUs," Oct 2018. [Online]. Available: https://github.com/jrowberg/i2cdevlib

[8] The MathWorks, Inc., "lqrd: Design discrete linear-quadratic (LQ) regulator for continuous plant," 2018. [Online]. Available: https://uk.mathworks.com/help/releases/R2018b/control/ref/lqrd.html?s_tid=doc_ta

[9] M. Harris, "DJI and Flyability partner to bring collision-tolerance to UAVs," Jun 2016. [Online]. Available: https://www.dji.com/uk/newsroom/news/dji-and-flyability-partner-to-bring-collision-tolerance-to-uavs?clickaid=4fldJQtFqsjsJmA5SKnt0E3EKnmH4Qms&clickpid=939637&clicksid=11b5e9336c911ebcee3d518dfaf37780

Appendix: Code Repository

The source codes used in this project can be found in the below link:

`https://github.com/ImperialCollegeLondon/QTIP`