|  |
| --- |
| Group 6A |
| Trajectory Tracking of a UAV |
| 49329 Control of Mechatronic Systems SPR 2021 |

|  |
| --- |
| Cameron Chan 13208487 (Leader)  Calvin Suryo 14096073  Chamath Edirisinhege 12977866  25/10/2021 |

Contents

[1 Problem Definition 1](#_Toc86076597)

[2 System Model 1](#_Toc86076598)

[3 Methodology 2](#_Toc86076599)

[3.1 Non-Linear Model 2](#_Toc86076600)

[3.2 MPC 2](#_Toc86076601)

[3.2.1. Cost Minimalization Function. 2](#_Toc86076602)

[3.2.2. Constraint Function 3](#_Toc86076603)

[3.3 Trajectory Generation 4](#_Toc86076604)

[3.4 Simulation 4](#_Toc86076605)

[4 Results 5](#_Toc86076606)

[4.1 MPC 5](#_Toc86076607)

[4.1.1 Example 1 – Pure Z 6](#_Toc86076608)

[4.1.2 Example 2 – Diagonal 8](#_Toc86076609)

[4.1.3 Example 3 – Circle XZ 10](#_Toc86076610)

[4.2 Simulink 11](#_Toc86076611)

[4.2.1 Random Noise 11](#_Toc86076612)

[4.2.2 Trajectory 1 – Pure Z with a different XY start 12](#_Toc86076613)

[4.2.3 Trajectory 2 – Diagonal XY with a different Z start 13](#_Toc86076614)

[4.2.4 Trajectory 3 – Circle XZ 15](#_Toc86076615)

[4.2.5 Trajectory 4 – Circle XY 16](#_Toc86076616)

[4.2.6 Trajectory 5 – Random Shape 18](#_Toc86076617)

[5 Future Work 20](#_Toc86076618)

[6 Conclusion 21](#_Toc86076619)

[7 References 22](#_Toc86076620)

# Problem Definition

Unmanned Aerial Vehicles (UAV) have been largely popular among all audiences during recent times. Commonly referred to as drones, are employed by the public and militaries for a wide variety of applications. Automated control systems are used to manage different aspects of UAVs depending on the purpose. Even though they can be remotely controlled by a pilot, dedicated control systems are used in most large or commercial UAVs. These unmanned vehicular platforms can be categorised into two main groups based on their flight mechanics, fixed-wing UAVs and fixed rotor UAVs.

This project is focused on the design and simulation of a control system for a UAV, which would improve its navigational capabilities by introducing the ability to follow desired trajectories. The control system would also consider the possible disturbances such as wind and other external conditions. In this project, we have based the kinematic model of the UAV on a quadrotor, fixed rotor UAV.

# System Model

There are a few different kinematic models for fixed rotor UAVs. The state-space representation and control method will vary greatly depending on the selected model. Aisuwarya et al. have an accurate model of a UAV and designed a PID controller for the model. This information is can be found in their paper. However, the goal of this project is to design a controller that allows the UAV to increase its flight capabilities by following pre-defined trajectories.

The 2019 paper by Tahir et al., provides several detailed state-space models for fixed rotor UAVs. The models in this paper were not specifically applied to specific control methods. These models used motor speeds as inputs rather than the states.

A standard quadcopter would have 6 degrees of freedom in motion as shown in figure 2.1 below. Those are x, y z, pitch (θ), roll (Փ), and yaw(ψ).

Chart, scatter chart

Description automatically generated

Figure 2.1: Visualization of the degrees of freedom in a standard fixed rotor UAV

The proposed model will omit yaw for simplicity. This report will focus on the calculations for 5 of the states which are x, y, z, pitch, and roll. The yaw of the model is considered always as 0.

Figure 2.2 below, shows the non-linear continuous equations on the left and the non-linear discretized equations on the right. The non-linear discretized equations are used in the simulation. To calculate the next position of translation and angle, the system uses these equations. The inputs of the system are the vertical speed and angular speed about the x and y axes.

Chart, scatter chart

Description automatically generated

Figure 2.2 Quadcopter Dynamics Equations

# Methodology

## Non-Linear Model

This report approaches the system using a non-linear model. This technique uses the equations that were displayed in the system model to calculate the next input that was most suitable to reach the desired location. To predict the most suitable input to reach the desired location our approach is by utilizing *Model Predictive Control* (MPC).

## MPC

Model Predictive Control is a model of the process to obtain the control signal by minimizing an objective function (Findeisen, 2017). Based on the description, its main components are the cost minimisation function (objective function) and the constraint function. The objective function contains the equation of how the system behaves and how to minimise it, and the constraint function constrains the calculations to the system limits. These limitations are important to make the system as real as possible because every real system always has limits such as maximum and minimum motor speeds.

## 3.2.1. Cost Minimalization Function.

As shown in figure 2, those are the kinematic equation of the system. So, in this system the states and the inputs are:

So, the equation of the objective function to be minimised is:

Where N is the prediction horizon, , , and Q ≥ 0, R ≥ 0 are weighting matrices for the error in the state and control variables, respectively. So, to get the minimum value of the function above this report uses “fmincon” MATLAB function.

MATLAB’s “fmincon” function is a non-linear programming solver that finds the minimum of a problem specified by:

Diagram

Description automatically generated

In the code, this function is called by this line of code:

[X,fval] = fmincon(@(X0)objfun(X0,N,X\_initial, X\_desired, U\_desired),X0,[],[],[],[],LB,UB,@(X0)confun(X0,X\_initial,N),options);

* X0 is the vector that the optimisation calculations are iterated and performed upon.
* N is the prediction horizon. A larger prediction horizon significantly increases the number of computations that need to be made to optimise the problem.
* X initial is the initial position of the UAV.
* X and U desired are the pre-defined trajectory vectors.
* The object function (objfun) contains the function (see the previous page).
* The constraint function (confun) contains the calculation to compute the next pose.
* LB and UB are the lower bound and upper bounds of the states and inputs.
* The weighting matrices of the input and states the system will use these matrices:

## 3.2.2. Constraint Function

The constraint function is to specify the limits of the model.

In this project constraints that was used was for the states and inputs are:

## Trajectory Generation

This section is to generate the desired trajectory that the MPC will try to follow. The codes below will create the reference states and input, and MPC must try to reach those values even when it faces disturbances or start in a different starting point.

The trajectories in the quadcopter simulation are 3 dimensional. The trajectories that will be simulated are described as equations below:

1. Pure Z translation
2. Diagonal XY Simulation
3. Circle XZ
4. Circle XY

1. Random Trajectory

These trajectories will be created by running the *trajectory.m*. Change the value of “traj“ to select the desired trajectory to simulate.

## Simulation

To apply the theories above, this project converts the program above into a MATLAB program and Simulink model. The main components that are available in these models are:

* *Trajectory.m*: This line of code main purpose is to create the desired states and the desired input as explained in the “Trajectory Generation” section. The output of this line of code is the value and and over time, stored in the workspace as X\_desired and U\_desired respectively.
* *objfun.m:*  This line of code contains the cost minimalization function that had been discussed previously.
* *confun.m:*  This line of code is used to create the next state according to the optimum input that the object function created.
* *compute\_next\_pose.m:* This code contains the non-linear, discretized equation of the system described in figure 2.2. This function is used in the confun.m.
* *quadcopter\_mpc\_function(X\_initial, N, X desired, U\_desired)* : This function is designed to be the complete MPC function that contains the objfunc.m and confun.m. This function also contains the constraints for the states and the inputs.

For the Simulink model the control of the quadcopter uses this model:

Diagram

Description automatically generated

Figure 3. Quadcopter Simulink Model

Noise is added to the states of the quadcopter via the addNoise function. The addNoise function creates a random number and multiplies it by a constant before adding it to the states.

# Results

## MPC

The MPC controller takes state limits, input limits, an initial state, a desired trajectory, and prediction horizon as inputs. It uses MATLAB’s “fmincon” function to calculate the most optimal vector of states and inputs.

The MPC controller can be tested using the “launch.m” file. This file plots the actual (calculated) states against the desired states.

### Example 1 – Pure Z

In table 1 below, each state [x, y, z, phi, theta] is represented by a row. The first column represents the initial desired state. The following columns represent the next N desired states, where N = 10   
(N = prediction horizon). The time between states is 1 second.

A prediction horizon, N, of 10 was selected for these examples to show that the states calculated by the “fmincon” function will eventually converge with the desired trajectory states.

Example 1 is a pure Z translation at a velocity of 0.3 m/s.

Table 4.1 MPC Example 1 - X\_desired

A screenshot of a computer

Description automatically generated with medium confidence

|  |  |
| --- | --- |
| 3D View | XY View |
|  |  |
| XZ View | YZ View |
|  |  |

Figure 4.1 MPC Example 1 - Initial Position [-5, -5, 0, 0, 0]

Figure 4.1 above, plots the actual and desired states when X\_initial = [-5,-5,0,0,0]. The plot shows that the most optimal series of inputs and states that will move towards the desired trajectory even when not starting from the desired initial position of [0,0,0,0,0].

In Figure 4.1, the blue markers are a visualisation of the initial plus 10 optimised states. However, only the first optimised input is outputted by the MPC controller. In this case, the input was:

U =

0.8822

1.5100

1.5100

Where U = [velocity, phi\_dot, theta\_dot] and is limited by its lower and upper bounds.

Chart, line chart, scatter chart

Description automatically generated

Figure 4.2 MPC Example 1 - Initial Position = Desired Initial Position

Figure 4.2 above shows that the MPC controller can output the necessary control inputs to follow a desired trajectory when the actual initial position and desired initial positions are the same.

|  |  |
| --- | --- |
| 3D View | XY View |
|  |  |
| XZ View | YZ View |
|  |  |

Figure 4.3 MPC Example 1 - Initial Position [-3, 0, 5, 0, 0]

Figure 4.3 plots the actual and desired states when X\_initial = [-3,0,5,0,0]. This shows that the MPC controller can move towards the desired trajectory from a different initial position that is ahead of the desired trajectory.

### Example 2 – Diagonal

Table 4.2 MPC Example 2 - X\_desired

A picture containing text, cabinet, screenshot

Description automatically generated

|  |  |
| --- | --- |
| 3D View | XY View |
|  | Chart, scatter chart  Description automatically generated |

|  |  |
| --- | --- |
| XZ View | YZ View |
|  |  |

Figure 4.4 MPC Example 2 - Initial Position [-5, -5, 0, 0, 0]

Figure 4.4 above, plots the actual and desired states when X\_initial = [-5,-5,0,0,0].

### Example 3 – Circle XZ

Table 4.3 MPC Example 3 - X\_desired

Table

Description automatically generated

|  |  |
| --- | --- |
| 3D View | XY View |
|  |  |

|  |  |
| --- | --- |
| XZ View | YZ View |
|  |  |

Figure 4.5 MPC Example 3 - Initial Position [-2, 0, -2, 0, 0]

Figure 4.5 above, plots the actual and desired states when X\_initial = [-2,0,-2,0,0].

## Simulink

The Simulink simulation takes 3 inputs:

* The prediction horizon: an integer between 1 and 10 (maximum can be changed).
* X desired: a generated set of states.
* U desired: a generated set of inputs.

To run the simulation, X desired and U desired first need to be generated by running “trajectory.m”. The Simulink simulation is set to run for 100s, the duration of the simulation should be equal to or less than the timestamps variable in “trajectory.m”. This variable is currently set to generate 100 desired states and inputs.

A larger prediction horizon will take a longer time to compute, a prediction horizon of 5 has been selected for the simulation results below.

The simulated trajectory always starts at [0,0,0,0,0]. The figures below show the desired trajectory on the left and the simulated trajectory on the right.

### Random Noise

Randomly generated noise is added to the states every 1 second. All the simulated trajectories above have noise added to the quadcopter states.

Chart

Description automatically generated

Figure 4.6 Noise Example

### Trajectory 1 – Pure Z with a different XY start

|  |  |
| --- | --- |
| Trajectory | Simulink |
| 3D View | |
|  |  |
| XY | |
|  |  |
| XZ | |
|  |  |

|  |  |
| --- | --- |
| YZ | |
|  |  |

Figure 4.7 Trajectory 1

The trajectory in figure 4.7 is a pure z translation from x = 5, y = 5. The quadcopter is able follow the trajectory when starting from [0,0,0,0,0].

### Trajectory 2 – Diagonal XY with a different Z start

|  |  |
| --- | --- |
| Trajectory | Simulink |
| 3D View | |
|  |  |
| XY | |
|  |  |

|  |  |
| --- | --- |
| XZ | |
|  |  |
| YZ | |
|  |  |

Figure 4.8 Trajectory 2

The trajectory in figure 4.8 is an x-y translation with z = 10. This shows that the quadcopter can follow the trajectory when starting from a different z position.

### Trajectory 3 – Circle XZ

|  |  |
| --- | --- |
| Trajectory | Simulink |
| 3D View | |
|  |  |
| XY | |
|  |  |
| XZ | |
|  |  |

|  |  |
| --- | --- |
| YZ | |
|  |  |

Figure 4.9 Trajectory 3

The trajectory in figure 4.9 is a circle on the XZ plane with y = 10 and a centre of x = 0, z = 20. The quadcopter is able follow the trajectory when starting from [0,0,0,0,0].

The quadcopter moves more in the y-direction rather than following the circle trajectory initially because that movement is more optimal in bringing the quadcopter closer to the desired trajectory. Once the quadcopter y = 10, then the quadcopter follows the circle trajectory closely.

### Trajectory 4 – Circle XY

|  |  |
| --- | --- |
| Trajectory | Simulink |
| 3D View | |
|  |  |

|  |  |
| --- | --- |
| XY | |
|  |  |
| XZ | |
|  |  |
| YZ | |
|  |  |

Figure 4.10 Trajectory 4

The trajectory in figure 4.10 is a circle on the XY plane with z = 10 and a centre of x = 0, y = 10. The quadcopter is able follow the trajectory when starting from [0,0,0,0,0].

### Trajectory 5 – Random Shape

|  |  |
| --- | --- |
| Trajectory | Simulink |
| 3D View | |
|  |  |
| XY | |
|  |  |
| XZ | |
|  |  |

|  |  |
| --- | --- |
| YZ | |
|  |  |

Figure 4.11 Trajectory 5

The trajectory in figure 4.11 is a random shape. This demonstrates that the quadcopter can follow trajectories with tight and sharp turns.

# Future Work

The current model of the UAV and the MPC based controller can follow many different trajectories very closely, as discussed in the previous section, there are still some flaws with our model and its trajectory tracking. The first major problem with our model is that we have ignored gravity, which causes the UAV to tilt unnaturally and travel with impossible orientations. Secondly, we have assumed a constant yaw angle of zero degrees, eliminating one degree of freedom, this is another major improvement we wish to make to our model. Further changes we would like to make to our model include, changing the inputs to rotor speed, instead of velocity, pitch and roll angles, adding the variables Q and R as inputs for the Simulink model and testing with more complicated trajectories with larger noise applied to the inputs and controls. Additionally, we would also need to change the weighting of the controls to make the actual trajectory of the UAV smoother than it is now.

We have used the angles of pitch and roll, and velocity as the control parameters. Where the velocity in this would be the result of the thrust generated from the propellors, pushing the UAV vertically upwards (positive Z-direction). At first glance this seems accurate, however, while testing the controller with trajectories we realized that the UAV would directly point its positive z-axis in the direction of the desired trajectory. This motion is unnatural even though it follows the trajectory closely. We would have to change the control inputs in a way that would incorporate gravity to fix this problem. Using accelerations instead of velocity and absolute orientational positions as inputs. The current model also allows the UAV to change its orientation instantly, which is not accurate.

Including the ability to control the yaw angle is another thing that would need to be added to the model. This would lead to much smoother trajectories since the yaw allows the UAV to change its orientation in the X-Y plane without changing its position. Thus, trajectories that involve changing directions in the X-Y plane would be a lot smoother. Additionally, it would make the model a lot more realistic if we were to change the inputs to rotor speeds, this way we would be able to incorporate the above changes into the model all at once. It would be quite complicated, but we would be able to apply the controller to a real-life system this way.

Finally, optimisation for the Simulink model would also be a good addition to the project. Adding the variables Q and R as inputs in the Simulink model would make the processing faster. The current model’s performance significantly worsens as we increase the prediction horizon above five. It would be a great chance to test the model using a large prediction horizon. Increasing the prediction horizon would improve the trajectory, following the desired trajectory much closer than it does now.

# Conclusion

The focus of this project was to design and simulate a control system for a UAV, which would also consider external disturbances and still be able to follow the desired trajectory. As discussed, we have obtained good results where we have designed and simulated a UAV to be able to closely follow the desired trajectory. Even though there are many improvements that we think could be made to our model and simulation, as highlighted in the previous section, given the time limit and the learning curve we have accomplished the goals we set to achieve through this project.

Some of the objectives of this project were to perform trajectory tracking in 3-dimensional space using model predictive control. Additionally, we also wanted to have a much deeper understanding of the usage and application of MATLAB and Simulink, by using the software for the simulations. Furthermore, we aimed to dig deeper into the application of discretization for non-linear problems. Comparing what we have accomplished to what we wanted to achieve, we can agree that we have achieved every one of the objectives we set out to achieve. Even though the result was not as perfect as we wanted it to be, we have gained a significant amount of experience and knowledge about the discretization of nonlinear systems, MATLAB, Simulink and model predictive control.

The functional objectives of the project were to design and simulate the controller to be able to follow a 3-dimensional trajectory, generate a visual simulation and the controller to be able to handle large disturbances. Referring to the results discussed in this report, we can say that our MPC controller manages to follow the 3-dimensional trajectory closely despite the introduction of noise, the actual trajectory remains close to the desired trajectory. This can be seen from the results of the simulation.

Overall, we can agree that both the learning objectives and the deliverables of the project have been met and proven with the results. There are flaws in our model which we have discovered during the testing process and identified future improvements we can make. We have achieved what we wanted to learn and more from this project.

# References

Aisuwarya, R., Yonas, F. M., & Yendri, D. Design of Autonomous Quadcopter Using Orientation Sensor with Variations in Load Fulcrum Point. *Lontar Komput. J. Ilm. Teknol. Inf*, 84.

Tahir, Z., Tahir, W., & Liaqat, S. A. (2019). State space system modelling of a quad copter UAV. *arXiv preprint arXiv:1908.07401*.

Findeisen, R., Allgöwer, F., & Biegler, L. T. (2007). Assessment and future directions of nonlinear model predictive control. . https://doi.org/10.1007/978-3-540-72699-9