Table of Contents

[Steps for Connecting to Drone to Bluetooth (Detailed Steps Link HERE) 2](#_Toc17974522)

[Steps For Running Simulation and Deploying Model to Drone 2](#_Toc17974523)

[Parrot Drone Quadcopter Model 3](#_Toc17974524)

[1. Overview of Quadcopter 3](#_Toc17974525)

[2. Overview of Quadcopter Model 4](#_Toc17974526)

[3. Command Sub-System 4](#_Toc17974527)

[4. Sensor Sub-System 5](#_Toc17974528)

[5. Flight Control System (FCS) 6](#_Toc17974529)

[5.1 Image Processing System 6](#_Toc17974530)

[5.2 Landing Logic Block 6](#_Toc17974531)

[5.3 Controller Block – Landing and Altitude Controller 8](#_Toc17974532)

[5.4 Controller Block – Starting Motors Block (For Syncing With Vicon) 9](#_Toc17974533)

[5.5 Controller Block – Yaw, XY Reference, Attitude, gravity feedforward blocks (For Syncing With Vicon) 10](#_Toc17974534)

[6. Flight Control Interface and Simulink Data Logging (RSData.mat) 11](#_Toc17974535)

[Possible Errors During Building/Deploying Model 13](#_Toc17974536)

[Links to Helpful Videos and Matlab Pages 14](#_Toc17974537)

[a) Help With Understanding Project Setup and Deploying Model 14](#_Toc17974538)

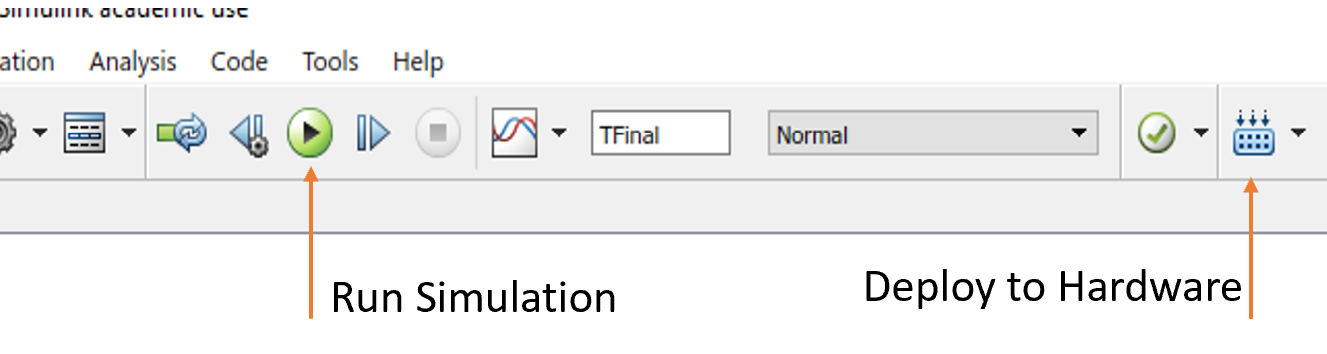
[b) Help With Common Issues 14](#_Toc17974539)

# Steps for Connecting to Drone to Bluetooth ([Detailed Steps Link HERE](https://www.mathworks.com/help/supportpkg/parrot/setup-and-configuration.html))

1. Use Simulink Support Package for Parrot Minidrones to update firmware on the drone. Open Add-on Manager on Matlab, select “Setup” on Simulink support package and follow instructions.
2. Connect the Bluetooth dongle and install the required drivers (CSR\_Harmony\_2.1.63.0). Saved on the computer in “Parrot Mambo” folder in E drive (or use this link to [download](http://47.88.26.219:81/Driver/CSR%20Harmony/)).
3. Go to This PC/My Bluetooth Devices and search for “ALL” devices. Parrot drone in either Mambo\_xxxxx or it Bluetooth ID will appear as a Gamepad (X-box like icon). NOTE: there will be two parrot drone devices that are detected. One will be a “X-Box” like gamepad, another a CPU-like box. Select the X-Box gamepad. If the gamepad does not show up, keep re-scanning until it appears. CONNECTING TO THE OTHER (CPU-like box) WILL NOT WORK.
4. Once Bluetooth connection is successful, right click the device (Mambo\_xxxx) and “Open Services”. Select “Personal Area Networking (NAP)” and connect. If there are no services (NAP does not appear), try restarting the computer, re-installing the drivers and disabling the inbuild Bluetooth drivers.
5. **IMPORTANT STEP:** if you have connected to Bluetooth and NAP successfully, DO NOT EVER, delete the paired device. It takes a long time to get it to detect the drone and NAP sometimes.

# Steps For Running Simulation and Deploying Model to Drone

1. Open the model.
2. Choose the “trajectory input method” from the “Command sub-system” and copy paste it into the “Input Trajectory” block in FCS. Depending on your chosen method, follow the steps below:
   1. **Signal Editor** – open the signal builder (Position/Altitude Reference) and launch signal editor. Build your desired signal and save.
   2. **Joystick** – connect a compatible joystick to the computer on a usb port. **NOTE:** Parrot Flypad goes to sleep after 2 minutes of connection time. You need to unplug and re-plug it to the computer every 2 minutes to use it.
   3. **Data** – use data save in a .mat file as the position and attitude data. File named “cmdData.mat” located in the “Main Models” folder of the project.
   4. **Excel** - use data save in a .xls file as the position and attitude data. File named “cmdData.xls” located in the “Main Models” folder of the project.
3. If using joystick model, go into **FCS/controller/Flight Controller/Altitude Switch** and change the Threshold of the switch to “**0**”. For any other input model set Threshold to “**TFinal**”.
4. Once ready, depending on whether running in Normal or External mode, deploy code as follows:
   1. **Normal Mode**: select “Deploy to Hardware” button. The code will be build, sent to the drone and a Flight Control Interface will pop up. Set power gain to 100% and run. **NOTE:** power gain is the percent of the maximum power provided by the batteries to the motors. 100% allows the motor speeds to reach maximum values while lower values (e.g 40%) will not be enough to even lift the drone.



* 1. **External mode**: select “Run Simulation” button. Next steps similar to Normal mode.

1. **NOTE:** to start motors use “**s or S**”. To land the drone use “**l or L**”. These require you to “Enable Keyboard Read” on the Flight Control Interface.
2. After flight is completed, click “MAT File” to save flight log in a .mat file and click “Flight Log” to save as a text file. Both of these are stored in the current Matlab running directory.

# Parrot Drone Quadcopter Model

## Overview of Quadcopter

**Axis**

The quadcopter body axis is centered in the center of gravity.

* The *x*-axis starts at the center of gravity and points in the direction along the nose of the quadcopter.
* The *y*-axis starts at the center of gravity and points to the right of the quadcopter.
* The *z*-axis starts at the center of gravity and points downward from the quadcopter

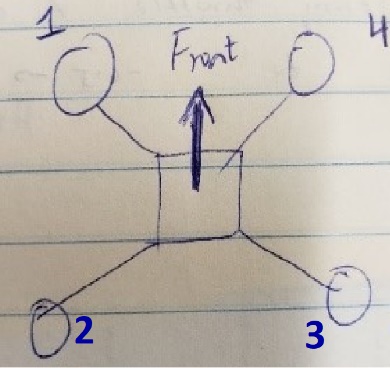


**Mass and Inertia**

We assume that the whole body works as a particle. The file “*vehicleVars*” contains the values for the inertia and mass.

**Rotors**

* Rotor #1 rotates positively with respect to the *z*-axis. It is located -45 degrees from the *x*-axis.
* Rotor #2 rotates negatively with respect to the body's *z*-axis. It is located -135 degrees from the *x*-axis.
* Rotor #3 has the same rotation direction as rotor #1. It is located 135 degrees from the *x*-axis.
* Rotor #4 has the same rotation direction as rotor #2. It is located 45 degrees from the *x*-axis.



## Overview of Quadcopter Model

A screenshot of a cell phone

Description automatically generated

The Quadcopter Model can perform both simulation and is able to be deployed onto a hardware for real world flights. Only the FCS (Flight Control System) block is deployed onto the drone and all other blocks (Command, Sensor, Environment, Airframe and Visualization) are for simulation purposes only. Changes were made to some of the blocks and are explained in detail in the following sections.

* Command Subsystem – Generates input as desired by the user to provide flight trajectory. **[Changes made to this subsystem].**
* Sensor Subsystem – Uses state estimates and environmental parameters to calculate the IMU and Gyroscope data. Also has a optical flow block that models the velocity (x,y,z) for simulation purposes and uses Image Processing using the downward facing camera for accurate positioning of the drone. **[Changes made to this subsystem].**
* Environment Subsystem – models the air pressure, density, temperature, speed of sound, gravity and magnetic field as either constant varying.
* Airframe Subsystem – uses a 6 DOF Quaternion (on the non-linear model) to output state estimates (velocity, acceleration, position on earth, e.t.c) of the drone required for simulation. A simplified linear model is available too.
* Visualization Subsystem – Used to create the 3D animated visual of the drone and surroundings to observe the drones flight when simulating.
* Flight Control System (FCS) – uses the user defined input, sensor readings and modeled drone dynamics (equations and PIDs) to output the required motor speeds to each motor enabling the drone to follow the user defined input. **[Changes made to this subsystem].**
* Flags – compares optical flow errors and geofencing errors to the estimated X and Y positions to trigger immediate drone shutdown if conditions are met.

## Command Sub-System

This subsystem provides 4 types of inputs positional and trajectory data that the drone follows during flight. Changes made in this model were made to the Joystick model only.

The Joystick model was designed to take yaw, pitch and roll commands from a connected joystick but the altitude was set fixed to the takeoff altitude set using a signal builder in the same Joystick block. To be able to take full control of the drone using all 4 motions (yaw, roll, pitch and thrust), the thrust output (that was initially terminated) used to output set the altitude of the drone as in the Figure below:

A screenshot of a cell phone

Description automatically generated

All other aspects of the Joystick model were maintained as before and the items in the “red block” were added. The signal output from “throttle cmd” ranged [0 to 1]. To increase altitude, we need a negative signal value (Z-axis orientation downwards) and to decrease altitude we need positive signal values. 0.5 is subtracted to change the “throttle cmd” signal range to [-0.5 to 0.5] to account for increasing and decreasing altitude. The integral (1/s) is to hold altitude as it will sum the current signal to the previous signal. Gain of -1 is to account for Z-axis orientation. The signal is outputted as a second output from the Joystick model as adding it to the “Command Bus” led to signal dimension and signal data type errors. The “throttle cmd” output is carried into the FCS.

## Sensor Sub-System

In the sensor subsystem, there are two subsystems; one generating IMU data and one optical flow data. The optical flow data has a control algorithm that models the optical flow of the drone. This is for simulation purposes only and if the simulation is run using the default gain of 1, the drone is seen to lose control at a height of about 2m. This is because of the “Velocity to optical flow gain” as shown in the Figure below.

A picture containing object

Description automatically generated

To increase the height of the drone, one can play around with this gain value in “*SensorVars*” file, however, this model has this value set to 0.75 which allows the drone to fly to about 2.5m high. Below is the answer provided my Matlab and can be accessed [HERE](https://www.mathworks.com/matlabcentral/answers/457427-how-can-i-increase-the-hover-height-of-the-quadcopter-parrot-mambo-without-causing-instability).

**“** The optical flow subsystem (asbQuadcopter/Sensors/Sensors (Dynamics)/Sensor System/Camera) is a crude representation of the optical flow sensor onboard the Rolling Spider/Mambo. Changing the "velocity to optical flow" gain value (Sensors.velocityToOpticalFlowGain) from 1.0 to 0.75 will allow you to increase the altitude to 2.5m without the xy instabilities that were observed previously. You can continue to manipulate this value as the altitude changes. **”**

Also, the default model contains a block called “Camera Model” in the optical flow block under sensor subsystem. This block uses Computer Vision Toolbox to provide for accurate position of the drone by using the downward facing camera and view of the floor/terrain to maintain stable flight/position. Computer Vision Toolbox is not available on the lab computer (at least at this time) and thus this model was deleted. Also, since there are no major external (wind e.t.c) forces in the lab, this is not a major concern.

A screenshot of a cell phone

Description automatically generated

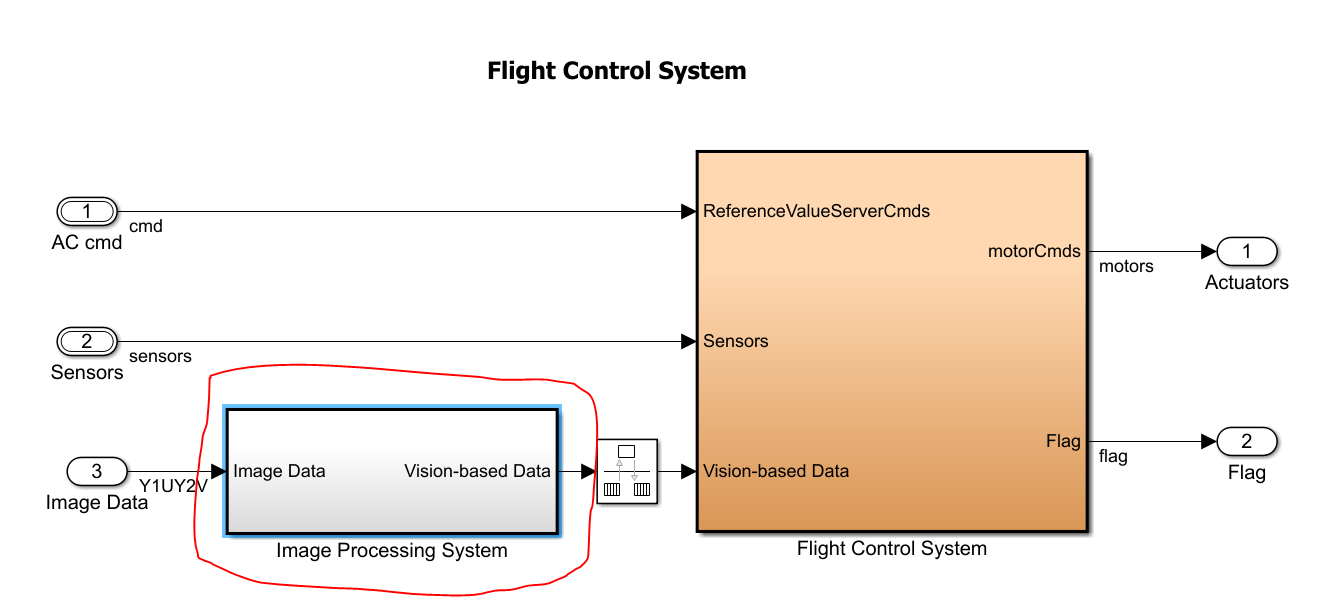
**NOTE:** The sensor block uses the state estimates and environmental factor from the non-linear airframe block to calculate the sensor outputs for simulation purposes as it cannot physically access the hardware on the drone during simulation. Thus, there is block that you will find in the sensor subsystem that actually accesses sensor data from the drone. It instead uses hardcoded initial position data alongside hardcoded sensor calibration values to calculate state estimates and output what might be the actual sensor signals on the drone hardware.

## Flight Control System (FCS)

This is the “Controller” of the quadcopter model. This is the subsystem that is loaded/deployed onto the drone enabling the drone to access sensor data, determine/calculate actuator outputs (motor speeds), compare to state estimates using filter (such as Kalman filters) and re-loop into the controller to form a closed loop model that is able to “fly the drone”.

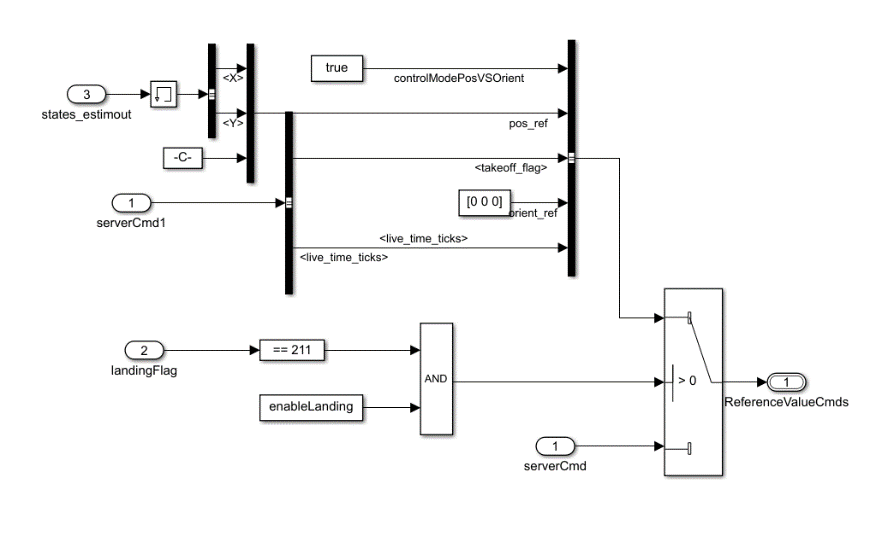
### Image Processing System

In FCS, there is a Image Processing System block that uses Computer Vision Toolbox and downward facing camera on the drone to determine the image colors and “noticeable” features of the floor that is uses to place itself stable during flight by looking for any changes in the image. Since Computer Vision Toolbox is not available, this too was deleted from FCS. It should be noted that this is not a model of the camera vision (as in the sensor block) but a block that actually accesses the camera data from the drone since FCS is deployed onto the drone hardware. Instead, we use input 3 as the throttle input from then “Command Subsystem”.



### Landing Logic Block

Next change in FCS was made to the “Landing Logic” block. As shown in the Figure below, this block used the state estimates and hardcoded orientation data to position the drone if a landing flag is triggered.



Landing flag is obtained from the Vision based data that we deleted in the previous step, so using the switch and landing flag as a switch trigger is not required and the reference commands can be directly used from input 1 (ReferenceValueServerCmds) of the FCS block.

If you do have access to “Computer Vision Toolbox”, leave the block above as it is but just copy paste your input method (signal builder or joystick) and replace the “ServerCmd” input as the output of the signal builder or joystick. Do as shown in the figure below.

A screenshot of a cell phone

Description automatically generated

We could either control the drone using pre-programmed data or in real time using “External Mode” in Simulink. To run External Mode, we need to have the input into the FCS as the Command Subsystem is not deployed/accessible by the drone. Thus, the input (ReferenceValueServerCmds) is terminated and the desired control model (i.e joystick, excel data or signal builder) is copied into the FCS in the block “Position and Orientation Input”. The output of the block is fed directly as a replacement of “ReferenceValueServerCmds” as in Figure below.

### A screenshot of a cell phone Description automatically generatedController Block – Landing and Altitude Controller

Inside the controller, few changes are made. A keyboard block is used to start the drone’s motors (“Starting Motors”) and also to provide a landing command (“Landing and Altitude Controller”). Inputs from the keyboard are read after “Start” is clicked on the Flight Control interface.

A screenshot of a cell phone

Description automatically generated

To enable keyboard reading, on the Flight Control Interface, click “Start”, then click “Show Keyboard Control Panel”, then click “Enable Keyboard Control”. Finally, the motors will not spin unless you press “s or S” on the keyboard. To land the drone, press “l or L”. (**MORE DETAILS BELOW**).

Also, to control the height of the drone using the throttle on the joystick (if joystick model is being used as the input), the “throttle cmd” signal that is outputted by the joystick is fed into the “controller” block as a 3rd input. The controller is where the attitude and altitude of the drone is calculated and used to determine the actuator (motor) speeds. As shown below, there is a section in the “Landing and Altitude Controller” block designated to determine an altitude command signal using the desired Z-value (in **RED**).

A screenshot of a cell phone

Description automatically generated

In the default model, “pos\_ref” signal is fed directly into the “gravity feedforward” block. A switch was added to make switching between the reference Z value and throttle easier. Since throttle is only needed if a joystick is being used, otherwise the altitude should be set by the reference Z-value from the signal builder/excel data, the switch is used. When using a Joystick, set the condition in the switch to “**u2~=0**” so that it will use the throttle to set the altitude of the drone. Using any other method, e.g signal builder, set the condition to “**u2>TFinal**”. Since TFinal is the simulation time, the altitude will be set by the signal builder for the full flight.

Also, a landing controller is added (in **BLUE** and **GREEN**). In the blue section, keyboard input “l or L” is detected (if pressed) and outputs the signal into a switch. The switch will output “zero” until “l or L” is pressed. Once it is pressed, the output signal will be “50” and will stay 50 forever. This will trigger the next switch reduce the current altitude of the drone by 0.1 meters every 1 second (or Ts\*0.1 = 0.0005m every 0.005s). You can adjust to whatever speed you want to land the drone at accordingly. When altitude reaches zero, the altitude is held at zero. Landing command (“l or L” from keyboard) will NOT work if a joystick is used and throttle is set as the Altitude output. The altitude is then sent to “gravity feedforward/equilibrium thrust” which outputs the appropriate motor thrusts.

### Controller Block – Starting Motors Block (For Syncing With Vicon)

For syncing Vicon and the drone, the motors will not run on the drone unless “s or S” is detected as a keyboard input. The “Starting Motors” block takes care of this. It reads the keyboard input, compares it to “s or S” and outputs a signal of value “100” if “s or S” is detected, as shown in the figure below.

A screenshot of a cell phone

Description automatically generated

It the holds the value 100 as the output until the flight/drone is stopped. So, once motors start, there is no way of stopping the motors unless you “STOP” it or “land” it using “l or L”. The motors are sent a command of [0 0 0 0] as motor signals from the moment the flight is started as shown in the figure below. This is under **FCS/Controller/thrustToMotorCommands.**

A screenshot of a cell phone

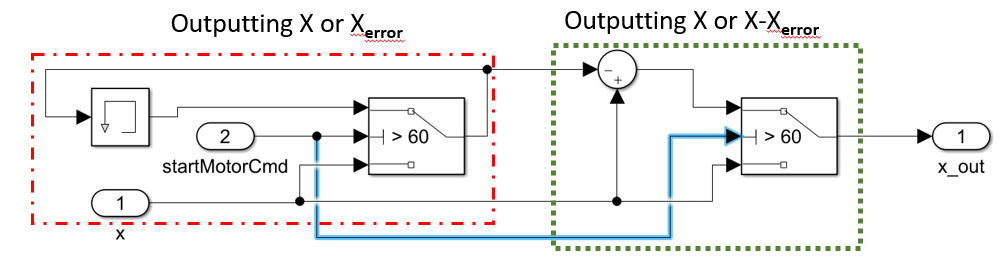
Description automatically generated

The switch trigger (“KeyboardRead” at input 2 of the switch) is the output from the keyboard block which detects if a “s or S” is pressed. Once it detects the “s or S”, the motors are sent the non-zero motor commands needed to fly the drone to meet the position and attitude reference at that moment in time.

### Controller Block – Yaw, XY Reference, Attitude, gravity feedforward blocks (For Syncing With Vicon)

To sync Vicon cameras with the drone data before flying the drone, it was needed that the drone be started and records data but does not have its motors spinning. The aim to syncing was to have both the drone and Vicon system record data but give the drone sudden movements manually by hand (move the drone around using your hands). Then, the drone is to be placed on the ground and motors are started using keyboard (s or S input as mentioned previously). In doing this, since the drone fixes the initial position (the moment you click START of flight control interface) as “zero”, and it is unlikely you will be able to precisely place it exactly back there after giving it random movements, the change is position and orientation needed to be accounted for.

Running trials without moving the drone showed that the initial state estimates were zero or nearly zero (around \*10-6). Thus a logic was implemented to reset the position, orientation, linear and angular acceleration to zero once the user has finished manual random movements and pressed “s or S” to start the motors. The figure below shows the logic:



For the example above, the logic takes as input the “X” state estimate from the state estimator block in FCS. The red block detects if the motor start command (keyboard input s or S) is inputted. If the motors are off, the first switch will keep on outputting the X estimate as determined by the state estimator (this will allow you to record the appropriate data when syncing with Vicon). If the motors are started, it will output the previous X position stored in the memory block (let’s call it Xerror). This will be the X position where you place the drone after making random movements manually. Our aim is to make the current position as reference zero. To do this, in the green block, any future X position estimate by the state estimator will has Xerror subtracted from it and will output the X position with reference set at Xerror as zero. Otherwise, if the motors are not yet started, it will output the X position as referenced from the initial zero position (at the time you click start on the flight control interface).

A similar change was made to the following variables; **X, Y, Z, dx, dy, dz, yaw, pitch, roll, p, q, r.**  The changes are in the following blocks of **FCS/Controller/Flight Controller,** as in the figure below (changes in **RED** blocks).

A screenshot of a cell phone

Description automatically generated

## Flight Control Interface and Simulink Data Logging (RSData.mat)

When “mat file logging” is enabled in the model’s configuration, flight data is logged (saved) into a “.mat” file that can be accessed after completion of flight. This file is stored in the running directory (current Matlab folder) by clicking “MAT File” in the flight control interface. A file called “RSData.mat” is saved. This RSData file stores a variety of data including sensor data, position data, motor signals among others, as shown in the figure below. The following describes the data stored in each structure in the RSData.mat file.

* rt\_motor – saves the final motor commands sent to all four motors. Motor numbering as follows:
* rt\_posref – saves the position reference values set as the input position reference values in order: [X, Y, Z, Yaw, Pitch, Roll]
* rt\_estim – state estimator output, in order: [X, Y, Z, Yaw, Pitch, Roll, dx, dy, dz, p, q, r].

p,q,r – angular acceleration.

* rt\_cmd – input commands (signal builder values).
* rt\_ optical – optical sensor/ camera sensor data
* rt\_calib – sensor calibration data
* rt\_sensor – data measured my IMU and ultrasound sensor. In order [ddx, ddy, ddz, p, q, r, altitude, prs, battery voltage, battery percent]

A screenshot of a cell phone

Description automatically generated

# Possible Errors During Building/Deploying Model

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Errors** | **Reason** | **Potential Solution/Web Links** |
| **1.** | Unable to check out the Matlab coder license which is needed to generate code | When user tries to use a function in an installed product but corresponding licence couldn't be found. | In case it is a shared licence Matlab is unable to connect to LM server for checking out a licence or there are too many users checking out the product at the time of licence checkout failure. |
| **2.** | Bluetooth Device (Gamepad icon) not found. | I have no idea why!! | Just keep searching. Remove the Bluetooth dongle and reinsert. Restart the drone. |
| **3.** | Cannot find “Personal Area Network (NAP)” service | I have no idea why!! | First try disabling internal Bluetooth drivers and restarting the computer. Reinstalling the drivers also helps. |
| **4.** | Error. Current model configurations differs from the model used to generate the utility code folder | Some setting in your current model are different from that used to generate the code. | Click “View differences” and configurations tab will open. Change the parameters that lead to the issue. All differences mentioned are in the configurations tab. Also, building code as mentioned at the start of this report will help. (“Open as top model”). |
| **5.** | Drone losses control for height above 2m. | Due to the optical flow being a crude model of Parrot drone and not the actual representation. | Change “Velocity to Optical Flow Gain” in Sensor/optical flow. |
| **6.** | Build procedure for model: XXX aborted due to an error | N/A | Delete content of work folder and re-build model. |
| **7.** | Failed to generate all binary outputs | Usually if you have “space” between word in folders. Example: “asb Quadcopter” as the main folder name or project name or model name will result in the error. “asbQuadcopter” is what is should be. | Avoid “space” when naming folders, projects, models or any file required for the project. Use a naming convention like: “XYZ” or “OneTwoThree” ad not  “X Y Z” or “One Two Three”.  Deleting contents of “Work” folder and rebuilding also helps. |

# Links to Helpful Videos and Matlab Pages

### Help With Understanding Project Setup and Deploying Model

* [Setting up control problem](https://www.mathworks.com/videos/drone-simulation-and-control-part-1-setting-up-the-control-problem-1539323440930.html)
* [Getting drone to hover](https://www.mathworks.com/videos/drone-simulation-and-control-part-2-how-do-you-get-a-drone-to-hover--1539323448303.html)
* [Building flight code](https://www.mathworks.com/videos/drone-simulation-and-control-part-3-how-to-build-the-flight-code-1539323453258.html)
* [Building model for simulation](https://www.mathworks.com/videos/drone-simulation-and-control-part-4-how-to-build-a-model-for-simulation-1539585112546.html)
* [Tuning PID Controller](https://www.mathworks.com/videos/drone-simulation-and-control-part-5-tuning-the-pid-controller-1540450868204.html)
* [Design, Simulate and Deploy Flight Controller for Parrot Drones](https://www.mathworks.com/videos/programming-drones-with-simulink-1513024653640.html) – shows how to run in both “Normal Mode” and “External Mode”. Also, to correctly build and deploy the **FCS ONLY**, right click FCS and **“Open as top model”** and deploy this to the drone.

### Help With Common Issues

* [Matlab and Simulink Support Packages for Parrot Minidrones](https://www.mathworks.com/matlabcentral/answers/457088-what-are-the-capabilities-of-the-matlab-support-package-for-parrot-drones-how-is-it-different-from?s_tid=answers_rc1-2_p2_MLT) - The Simulink Support Package for Parrot minidrones lets you build and deploy flight control algorithms on Parrot minidrones and works with a custom Parrot Firmware. This lets you deploy algorithms wirelessly over Bluetooth®. The MATLAB support package uses Wifi connection over BLE. This support package works with the default factory shipped firmware of the drone. R2019a Matlab supports only Parrot Mambo FPV.
* [Matlab and Flypad Compatible Firmware](https://www.mathworks.com/matlabcentral/answers/460529-control-the-parrot-mambo-with-the-simulink-compatible-firmware-through-the-flypad-phone?s_tid=answers_rc1-2_p2_Topic) - The Parrot Flypad works only with a drone having the factory shipped firmware. To work with the custom firmware, you can use the keyboard control/Joystick/Excel/Signal Builder block to manually control the drone while reading sensor data in Simulink using the Simulink Support Package for Parrot minidrones. To work with the Parrot Flypad directly, the MATLAB Support package for Parrot drones works with the factory firmware and lets you read navigation data, control the drone, and capture images/video from the drone's FPV camera. This should also work in parallel with the Parrot Flypad that ships with the drone but ONLY for Parrot FPV Drone.
* [Using Flight Control Interface](https://www.mathworks.com/help/supportpkg/parrot/ug/use-flightcontrol-ui-on-parrot-minidrone.html) – Matlab command: Parrot\_FlightInterface