Getting started with MIT's Rolling Spider MATLAB Toolbox

with Parrot's Rolling Spider Drone!

An MIT take-home lab for
16.30 Feedback Control Systems

Prof. Sertac Karaman, Fabian Riether
October 1, 2015

Massachusetts Institute of Technology

MIT AEROASTRO

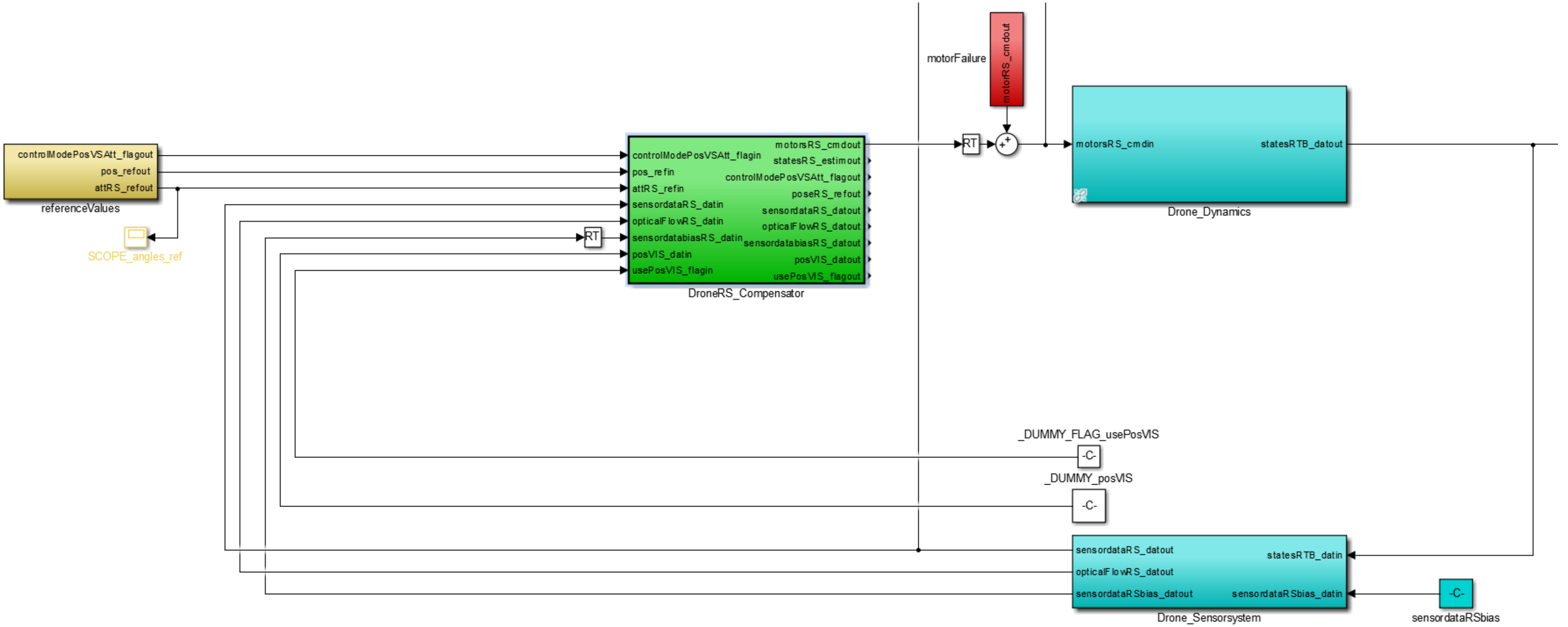
MIT's

Rolling Spider MATLAB Toolbox

with Parrot's Rolling Spider Drone!

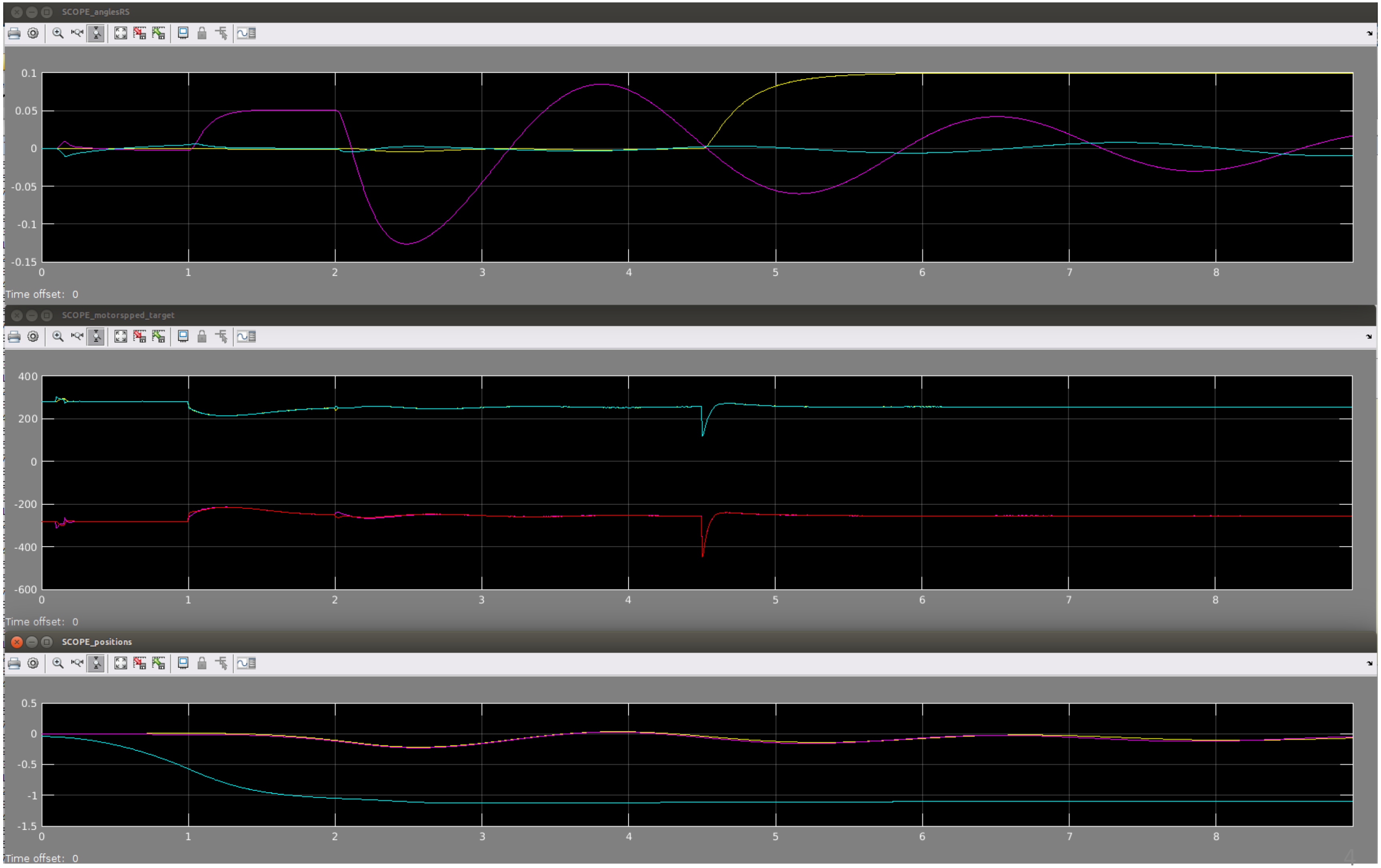
... let's you **design** and **simulate** estimation and control algorithms for a drone in MATLAB/Simulink and **autogenerates** embedded c-code that you can use to actually **fly** the drone! After your flight, recorded data can be **visualized** and analyzed.

Simulate ...

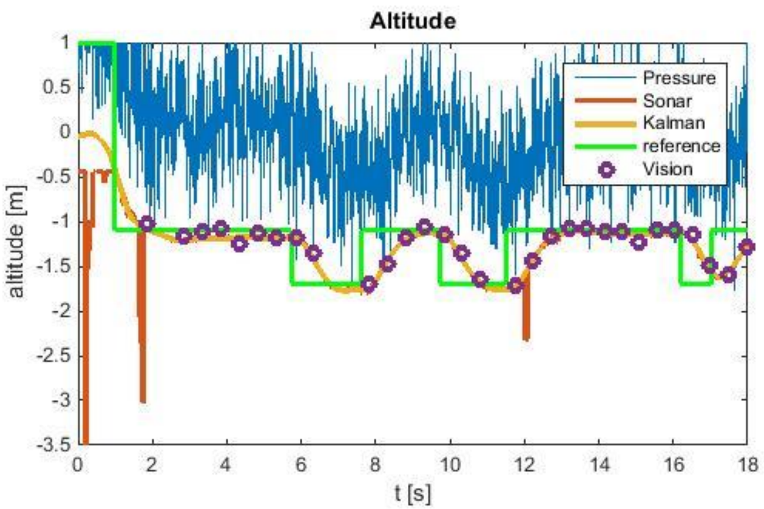


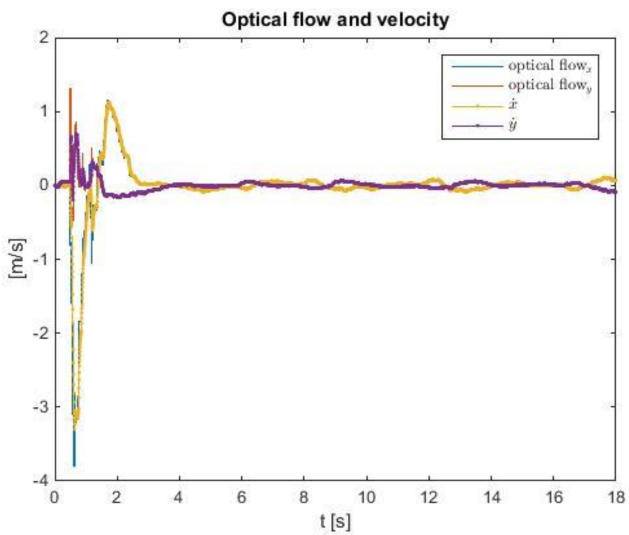
motorFailure
motorRS_cmdout
controlModePosVSAtt_flagout
pos_refout
attRS_refout
referenceValues
SCOPE_angles_ref
RT
controlModePosVSAtt_flagin
pos_refin
attRS_refin
sensordataRS_datin
opticalFlowRS_datin
sensordatabiasRS_datin
posVIS_datin
usePosVIS_flagin
motorsRS_cmdout
statesRS_estimout
controlModePosVSAtt_flagout
poseRS_refout
sensordataRS_datout
opticalFlowRS_datout
sensordatabiasRS_datout
posVIS_datout
usePosVIS_flagout
DroneRS_Compensator
RT
motorsRS_cmdin
statesRTB_datout
Drone_Dynamics
_DUMMY_FLAG_usePosVIS
-C-
_DUMMY_posVIS
-C-
sensordataRS_datout
opticalFlowRS_datout
sensordataRSbias_datout
statesRTB_datin
sensordataRSbias_datin
Drone_Sensorsystem
-C-
sensordataRSbias

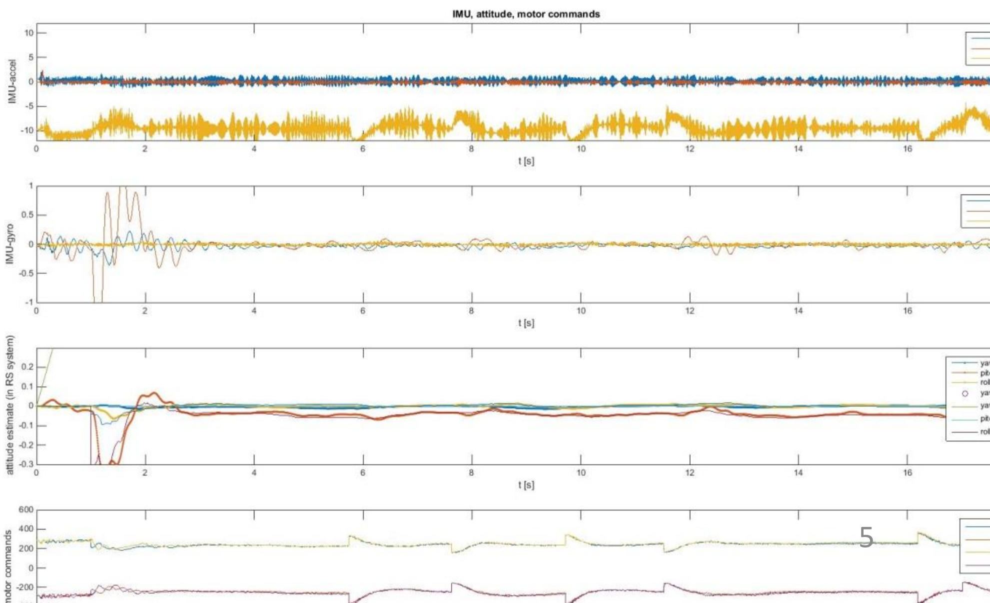
Plot simulated data



Plot recorded data







Contents

- Drone
  - Hardware
  - Safety
- Software interface *„How can we hack it?“*
- Toolbox *„What‘s the workflow?“*
  - Simulation and control design
  - Embedded code generation
  - Flying
  - Data Analysis
  - Dynamics Analysis
  - Beta-Feature: Vision

Drone-Hardware

- Parrot Rolling Spider (note: not the end2015-EVO version!)
  1. Mass: 68g
  2. Motors: 33g Thrust/motor
  3. embedded linux system
  4. IMU: 6axis-accelerometer-gyroscope
  5. Altitude sensors: Sonar, Pressure sensor
  6. Vision: Downward-facing camera, 160x120
  7. Battery: 7-8min flight time
- Bluetooth BLE adapter (if your laptop does not provide it, e.g. IOGEAR Bluetooth 4.0 USB Micro Adapter (GBU521))
- Safety goggles
- optional
  1. Additional battery and charger
  2. Extra set of propellers

Drone-Safety

Hardware Safety

- Stick to Parrot’s safety guidelines (print-out)
- Don’t charge batteries unattended
- Always fly with wheels installed

- Wear safety glasses all the time
- Only fly indoors, open area >10’x10’ for hover experiments
- Ensure people, animal, property, etc. safety
- Be smart!

- Always test a new program with 10% power first aka test run
- Stick to software safety procedures (p. 23)

Software Interface

„How do we hack it?“

- **Drone calls our control code @200Hz,**
  input: sensor data   output: motor commands
  `void RSEDU_control(HAL_acquisition_t* hal_sensors_data, HAL_command_t* hal_sensors_cmd)`

- **Drone calls our image processing code @60Hz**
  input: image buffer
  `void RSEDU_image_processing(void * buffer)`

FIFO

- **Drone calls our optical flow code @60Hz**
  input: computed optical flow
  `void RSEDU_optical_flow(float vx, float vy, float vz,int defined,float qualityIndicator)`

Massachusetts Institute of Technology

MIT AEROASTRO

Toolbox

„What's the workflow?"

MATLAB

1) Simulation and Control-Design

MATLAB

2) Embedded code generation

3) Controlcode integration into embedded system framework

Communication

Safety

4) Upload of shared library to Drone via Bluetooth

5) FLIGHT

5) Download of mat-file and images

6) Data analysis

7) Estimator-Control-***Re***design

Toolbox

A step-by-step

tutorial to guide you from

simulation to flight

Massachusetts Institute of Technology

MIT AEROASTRO

Toolbox

Workflow 0.1: Equipping your VirtualMachine/ubuntu (once)

- Let [ROSMAT] denote the path to the MIT toolbox root folder (i.e. the folder containing the README- and LICENSE file).
- If using the MIT 16.30 Feedback Control Systems virtual machine, you should only need to activate your MATLAB. Have your Mathworks student account ready (check MIT’s IST webpage for MATLAB) and open MATLAB via the desktop icon shortcut (you might have to run it from a terminal with `sudo matlab`). [ROSMAT] is ~/Downloads/RollindSpiderEdu-master/MIT_MatlabToolbox
- Your own ubuntu-system should be equipped with the following programs (if not, install them):
  1. MATLAB 2015a
  2. Lftp
     `sudo apt-get install lftp`
  3. Bluetooth stack
     `sudo apt-get install bluez-compat`
  4. expect
     `sudo apt-get install expect`

Massachusetts Institute of Technology

MIT AEROASTRO

Toolbox

Workflow 0.1: Equipping your VirtualMachine/ubuntu (once, cont.)

5. MIT's ROSMAT: Checkout https://github.com/Parrot-Developers/RollingSpiderEdu. Let [ROSMAT] be the path to the MIT-toolbox root folder containing the README- and LICENSE-file
6. Unpacked Gcc-arm-Toolchain in */opt/arm-2012.03/…* (files can be found in *[ROSMAT]/libs/gcc-arm-Toolchain*)
   - on 64bit systems, also install the following programs by entering in a terminal
   `sudo apt-get install lib32z1 lib32ncurses5 lib32bz2-1.0`
7. Add binaries folder to PATH: `sudo gedit ~/.profile`
   Append line, save, then lockout and in again.
   `export PATH=$PATH:[ROSMAT]/bin:[ROSMAT]/bin/utils:[ROSMAT]/bin/firmware`
8. Build utils
   `BuildUtils.sh`

Info on MATLAB toolboxes:
recommended to have: Communications System Toolbox, Computer Vision System Toolbox, Control System Toolbox, Embedded Coder, Fixed-Point Designer, MATLAB Coder, MATLAB Compiler, MATLAB Compiler SDK, Signal Processing Toolbox, Simulink, Simulink Coder, Simulink Control Design, Stateflow, Symbolic Math Toolbox
additionally part of MIT's 16.30 VirtualMachine for MIT students: Curve Fitting Toolbox, DSP System Toolbox, Fuzzy Logic Toolbox, Global Optimization Toolbox, Image Acquisition Toolbox, Image Processing Toolbox, Instrument Control Toolbox, MATLAB Report Generator, Model Predictive Control Toolbox, Optimization Toolbox, Robotics System Toolbox, Robust Control Toolbox, Simscape, Simulink 3D Animation, Simulink Design Optimization, Simulink Design Verifier, Simulink Report Generator, Simulink Verification and Validation, System Identification Toolbox, Vision HDL Toolbox, Wavelet Toolbox

Toolbox

Workflow 0.2: Flashing the drone (once)

The consumer drone has to be flashed with a custom firmware once.

1. Connect drone via USB (if using a virtual machine, make sure to connect to ubuntu)
2. Open *fvt6.txt* on drone USB, note down name and MAC address.
   (If no fvt6.txt can be found, skip step 3 for now and run `sudo hcitool scan` after step 9. Your drone should be listed, read the MAC address from there. Then do step 3 and continue with step 10 afterwards.)
3. Save MAC address to *DroneMACaddress.txt* by entering, in a terminal
   `DroneSetMACAddress.sh [MACADDRESS]`
4. Upload main firmware to drone by running
   `EDUfirmwareUploadSYS.sh`
   (Info: This script copies *rollingspider.edu.plf* to root folder of drone USB device)
5. Disconnect drone by ejecting USB device and removing USB cable
6. Charge battery
7. Insert battery
8. Wait until LEDs stopped blinking (firmware is now updated)
   (Note: If LEDs never blinked, redo step 1 & 3-8.)
9. Plug in bluetooth adapter (if necessary)

…continue on next slide!

Videotutorial: 01_FlashingTheDrone

Toolbox

Workflow 0.2: Flashing the drone (once) (cont)

10. Connect drone to computer by running
    `DroneConnect.sh`
11. Upload firmware files by running
    `EDUfirmwareUploadFILES.sh`
    (Info: uploads files in *[ROSMAT]/libs/EDUfirmwareFILES* to drone via ftp and IP 192.168.1.1)
12. Reboot drone
    `DroneReboot.sh`

…continue on next slide!

Videotutorial: 01_FlashingTheDrone

Toolbox

Workflow 0.2: Flashing the drone (once) (cont)

13. Connect drone again
    `DroneConnect.sh`
14. Initialize drone firmware
    `EDUfirmwareInitialize.sh`
    (Info: This script moves firmware files to right locations and grants permissions rights:
    ```
    mv /data/edu/dragon-prog /usr/bin/
    chmod +x /usr/bin/dragon-prog
    mv /data/edu/SpiderFlight.sh /bin/
    chmod +x /bin/SpiderFlight.sh)
    ```
15. Initialize drone
    `DroneInitialize.sh`
    (Info: This script write the computer's IP address to the drone's parameter file)
16. Done with flashing. Nice!

Videotutorial: 01_FlashingTheDrone

Toolbox

Workflow 0.3: (Dis-)Connecting to the drone (after restarts, ...)

- If you want to disconnect, run in a terminal
  `DroneDisconnect.sh`
- Connect via , run in a terminal
  `DroneConnect.sh`

Toolbox

Workflow I: Simulation and Control Design

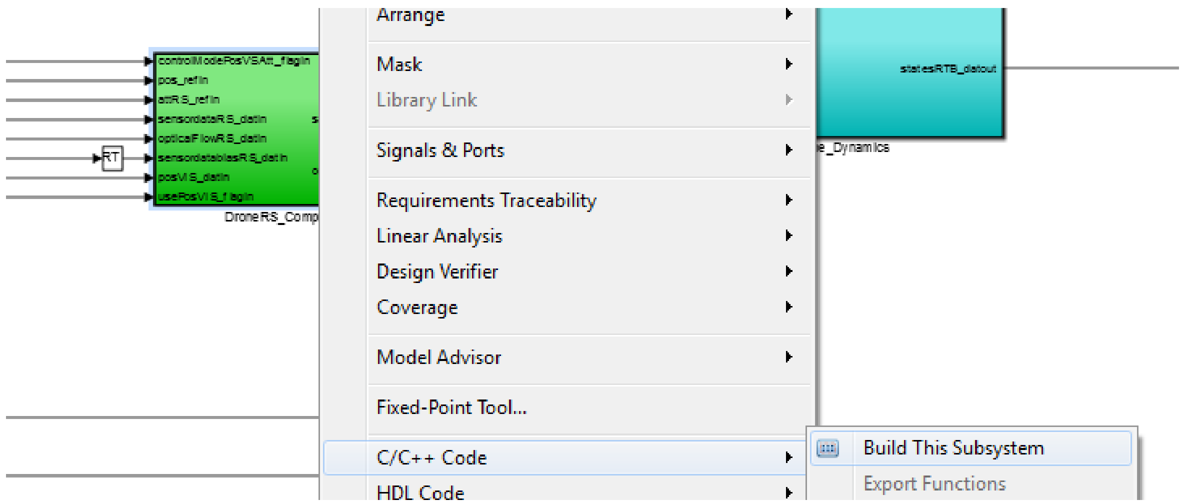
1. Open MATLAB and navigate to the `[ROSMAT]/trunk/matlab/` folder
   - `Simulation/` contains the Simulink files to design and simulate the drone with its estimators and controllers.
   - `libs/` contains parts of Peter Corke's Robotics Toolbox to simulate the dynamics of a drone, updated to (somewhat) match Parrot's Rolling Spider
   - `ExperimentAnalyzer/` contains various files to analyze sensor and dynamics data recorded while flying, or processing times from threads running on the drone.
2. Run `startup.m`, then open `sim_quadrotor.slx`
3. Design your controllers.
   As a first approach, copy-paste a preset controller: Open *controllers/controller_PID2W/controller_PID2W.slx*, copy the *ControllerPID2W* block and insert it at the correct place in *sim_quadrotor.slx*.
   For further design, Simulink can be used (mostly) freely, but keep in mind that c-code for a drone with low processing-power will be generated. (See *Notes_IdeasIssues* for more hints).
   Do not change the input/output-ports of the DronesRS_Compensator to avoid manual changes in the resulting c-code.
4. Open SCOPES to have variables plotted, press ▶ to simulate

Videotutorial: 02_DesigningControllers

Toolbox

Workflow II: Embedded code generation (1/2)

1. Rightclick on the *DronesRS_Compensator* block, select *„Build This Subsystem".*



In the pop-up dialogue box, click *Build.*

Videotutorial: 02_DesigningControllers

Toolbox

Workflow II: Embedded code generation (2/2)

2. Upload your controller
   (If disconnected from Drone: `DroneConnect.sh` first)
   `DroneUploadEmbeddedCode.sh`
   (Info: this script packs the autogenerated code with the drone's c-code framework using the binary `PackEmbeddedCode`, builds the code with make in *[ROSMAT]/trunk/embcode/build-arm* and uploads new shared library *[ROSMAT]/DroneExchange/librsedu.so* to drone using ftp to 192.168.1.1)

- *Expert level* - With changed Simulink input/output-ports:
  Do the steps from step 2 manually. After running PackEmbeddedCode in its folder, replace code paragraph "Input/Outputport Declarations IO(x)"…
  of SIMULINK compensator block in *rsEDU_control.c* with input/output-port declarations found in *ert_main.c*. Note: You also have to update function calls for initializing, stepping and packing the model in *rsedu_control.c*; found in *rsedu_control.c* with comments "IO(2)" and "IO(3)")

Videotutorial: 02_DesigningControllers

Toolbox

Workflow III: Flying (1) – Flight Phases

The drone’s flight is split into 3 phases

1. Sensor calibration
   It sits on the floor for 2 seconds to calibrate its sensors.

2. Take-off
   Take-off for 1 second with given power and attitude control only to about 1m altitude.

3. Actual flight

Massachusetts Institute of Technology

MIT AEROASTRO

Toolbox

Workflow III: Flying (2) – Safety Procedures

- Stick to Parrot's safety guidelines (see print-out)
- Always fly with wheels installed

- Wear safety glasses all the time
- Only fly indoors, open area >10'x10' for hover experiments
- Ensure people, animal, property, etc. safety
- Be smart!

- Always test a new program with *DroneTest.sh*, i.e. 10% power, first
- Stick to software safety procedures (p.23)

Toolbox

Workflow III: Flying (3) – Software Safety

- If the drone's main script does not crash itself, it shuts down the motors in case of a crash or a loss of optical flow
- A single flight is aborted automatically after 20 seconds

- For all other cases (and they will happen!)
- Always have a separate terminal open, enter `telnet 192.168.1.1` (you should already be connected to the drone), now you are logged directly onto the drone.
  Type
  `killall -s SIGKILL dragon-prog; gpio 39 -d ho 1;test-SIP6_pwm -S;`
  and be ready to execute this line when the drones goes crazy!

- After an automatic 20-seconds shutdown or a crash, shut down server manually by hitting 'e' in the server terminal (or Cntrl+C to kill it)

- Read through `~/ROSMAT/media/NOTES_ISSUESIDEAS`

Toolbox

Workflow III: Flying (4) – Settings

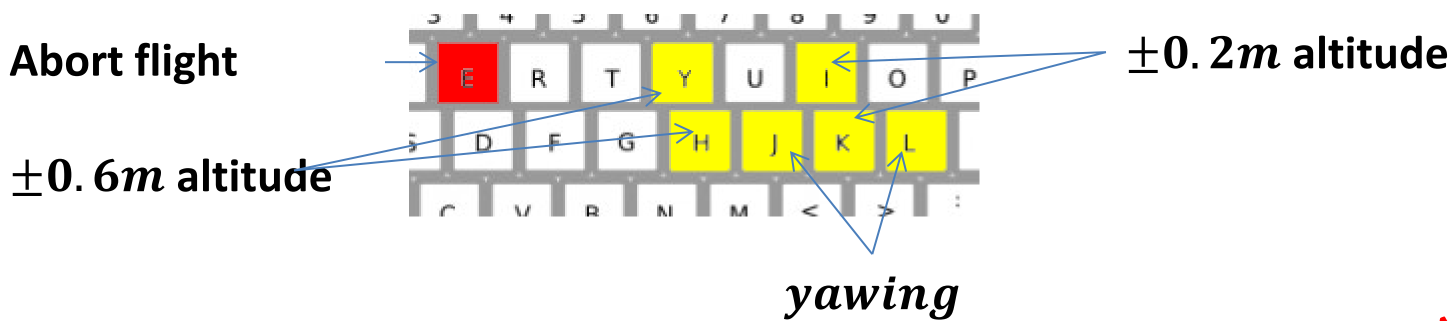
If you want to enable/disable software features

- In a new terminal, log onto drone `telnet 192.168.1.1`
  `vi /data/edu/params/paramsEDU.dat`
- Enable features with replacing ‘0’ by ‘1’
  FEAT_TIME: records timestamps for entering and leaving the functions rsedu_control, rsedu_of (optical flow) and rsedu_vis (visual position reconstruction)
  FEAT_OF_USE: optical flow is used to stabilize position
  FEAT_POSVIS_RUN: camera looks for landmark setup and reconstructs position if all landmarks found; visually reconstructed position is recorded
  FEAT_POSVIS_USE: use visually reconstructed position to enhance kalman position estimate
  FEAT_NOLOOK: compute color conversion, landmark matching etc. online instead of using a precomputed lookup-table (don’t use this, too slow)
  FEAT_IMSAVE:
  1: saves images
  2: images are being streamed to ubuntu machine (see rsedu_vis.c for additional instructions)
- FEAT_NOSAFETY: 1: drone is not automatically shut down when take off-surface is not level, z-axis – acceleration is positive or x-y-accelerations exceed 6m/s² (dangerous setting!)
- POWERGAIN cannot be changed manually

Toolbox

Workflow III: Flying (5)

1. Start KeyboardPilot, i.e. the server providing the reference values, with `DroneKeyboardPilot.sh`
2. In another terminal
   - `DroneTest.sh` for a test run with 10% Power
   - `DroneRun.sh` for a full run
3. Go back to the KeyboardPilot's terminal , hit keyboard buttons…



Videotutorial: 03_FlyingAndAnalyzing…

Toolbox

Workflow IV: Data Analysis – FlightAnalyzer

1. Download *RSdata.mat* from drone via ftp to *[ROSMAT]/DroneExchange/* by running `DroneDownloadFlightData.sh`
   (Alternatively, connect drone via usb and run `DroneDownloadFlightDataUSB.sh` (faster)
2. In MATLAB, load *RSdata.mat* (double-click)
3. Run MATLAB-script `FlightAnalyzer`

Videotutorial: 03_FlyingAndAnalyzing...

Toolbox

Workflow IV: Data Analysis – Software in the Loop

Instead of a full-stack simulation, feed *recorded* sensor data through the Simulink *DronesRS_Compensator* block to see what happened within the estimators and controllers during the recorded flight.

1. Make sure to have loaded some flight data *RSedu.mat* and have run the *FlightAnalyzer* once
2. In MATLAB, navigate to *[ROSMAT]/trunk/matlab/Simulation*
3. Use Simulink model *sim_SoftwareIntheLoop_Compensator.slx*

Toolbox

Workflow IV: Data Analysis – Processing Times

1. Download folder *ptimes/* from drone with `DroneDownloadPTimes.sh`
2. Run matlab script `ptimesAnalyzer`

Toolbox

Workflow V: Dynamics Analysis

- Check *[ROSMAT]/trunk/matlab/Simulation/controllers/controllers_fullstate* to see examples how to utilize MATLAB and Simulink to linearize dynamics

Toolbox

Info: Resetting the drone

- To fully reset the drone to its original state,
  - Do a software update via http://www.parrot.com/usa/support/parrot-rolling-spider/: Click on "Software Update“ and then "Download“, follow the instructions on the webpage.

Toolbox

Workflow VI: Betafeature-Vision

- Process recorded images: Converting recorded, binary images to jpg, save vision-inferred poses into pose.txt
  1. Download binary images from drone: `DroneDownloadImages.sh`
  2. Run `VisionPrePostProcessor.sh`
     Follow instructions
  3. See *[ROSMAT]/DataExchange/imgs/processed* for jpgs, poses, and landmark-identification images.

- Update landmark matrices:
  When using a different landmark setup, use *[ROSMAT]/trunk/VisionPrePostProcessor/findPostReconstructionParameters.m* to compute new vision matrices, then update them in *[ROSMAT]/trunk/embcode/rsedu_vis.c*