pdfauthor=, pdftitle=, pdfkeywords=, pdfsubject=, pdfcreator=Emacs 26.3 (Org mode 9.3.6), pdflang=English

Meeting Minutes

Teams
17 May, 2020

Address:
www.meetingnotes.no
Enterprise Number:

PARTICIPANTS	PRESENT	ABSENT	FOR YOUR INFORMATION
Abdullah Sherif - as394@hw.ac.uk	•		
Vishakh Kumar - vpk2@hw.ac.uk	•		
Vishnu Sarathy - vks2@hw.ac.uk	•		
Dr Mehdi Nazarinia			•
Dr Ityonna Amber	•		

PROJECT	DATE	CLASSIFICATION
keyword(PROJECT)		Unrestricted

1 Agenda

Agenda

- Housingkeeping & Onboarding
- Current Status of Project
- Action items

2 Housekeeping & Onboarding

2.1 New member!

Vishnu Sarathy is our new member! Introduce yourself!

2.2 Git Repository

Get used to git because that's the easiest way for us to work together. Great documentation at https://git-scm.com/doc

Quick reference:

```
git pull origin master
git add .
git commit -m "Useful message"
git push origin master

Our repo: https://gitlab.com/grokkingStuff/MathWorksMiniDrone
```

3 Current Status of Project

3.1 Mission Timeline

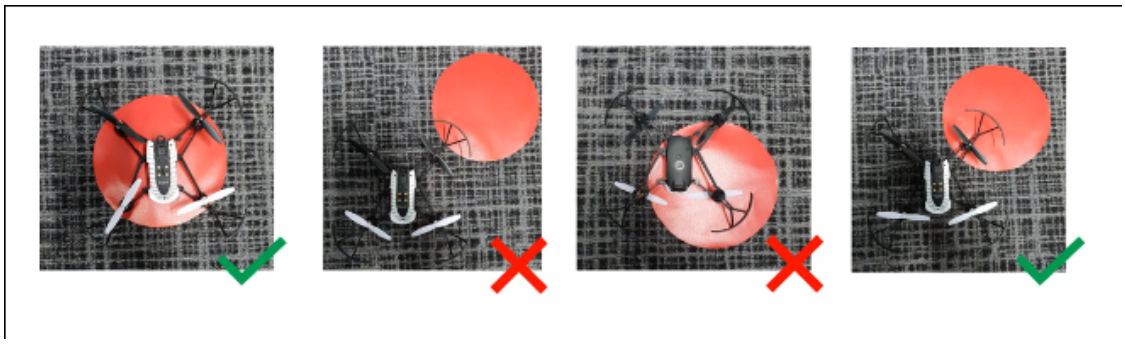
Key feedback:

- Test, test, test Compare the ROS-model ORB-SLAM vs the Simulink model IF it passes the stages, compare times. Work w/ Vishnu & Sheriff on this ASAP

3.1.1 Time consideration?

Time consideration? Time considered only when the minidrone lands on the end marker.
In order for the minidrone to be considered as having landed:

- Minidrone must be upright.
- Minidrone's bottom surface has to touch the floor.



3.2 General Software Stack

Work off ROS inside Simulink.

- Why?

Plenty of existing models in ROS that are industry-tested. Also, it's usually been optimized for the purpose and the ability to just subscribe to a node that gives you information is crazy useful

- So why Simulink?

The control logic is best expressed in Simulink because writing C code that has a decent state machine AND is able to interface with ROS nodes AND is able to help us out with a GUI/debug tools IS HARD! Simulink acts as a glue between our hardware, voodoo blackbox, and our ideas.

3.2.1 Matlab/Simulink

Not gonna write much here because I think everyone is familiar with it.
Status: Installed

3.2.2 ROS

Robot Operating System (ROS) is a framework of tools, libraries, and software to aid in robot software development. It is a flexible system for programming robots and controlling robotic platforms. ROS was developed by an open-source collaborative community to help grow the world of robotics. Applications for working with hardware, robotic simulation models, path planning, localization and mapping, and many other algorithms are available. For an introduction to ROS, see the ROS Introduction on their website.

1. ROS Toolbox in Simulink

   https://www.mathworks.com/help/ros/gs/robot-operating-system-ros.html

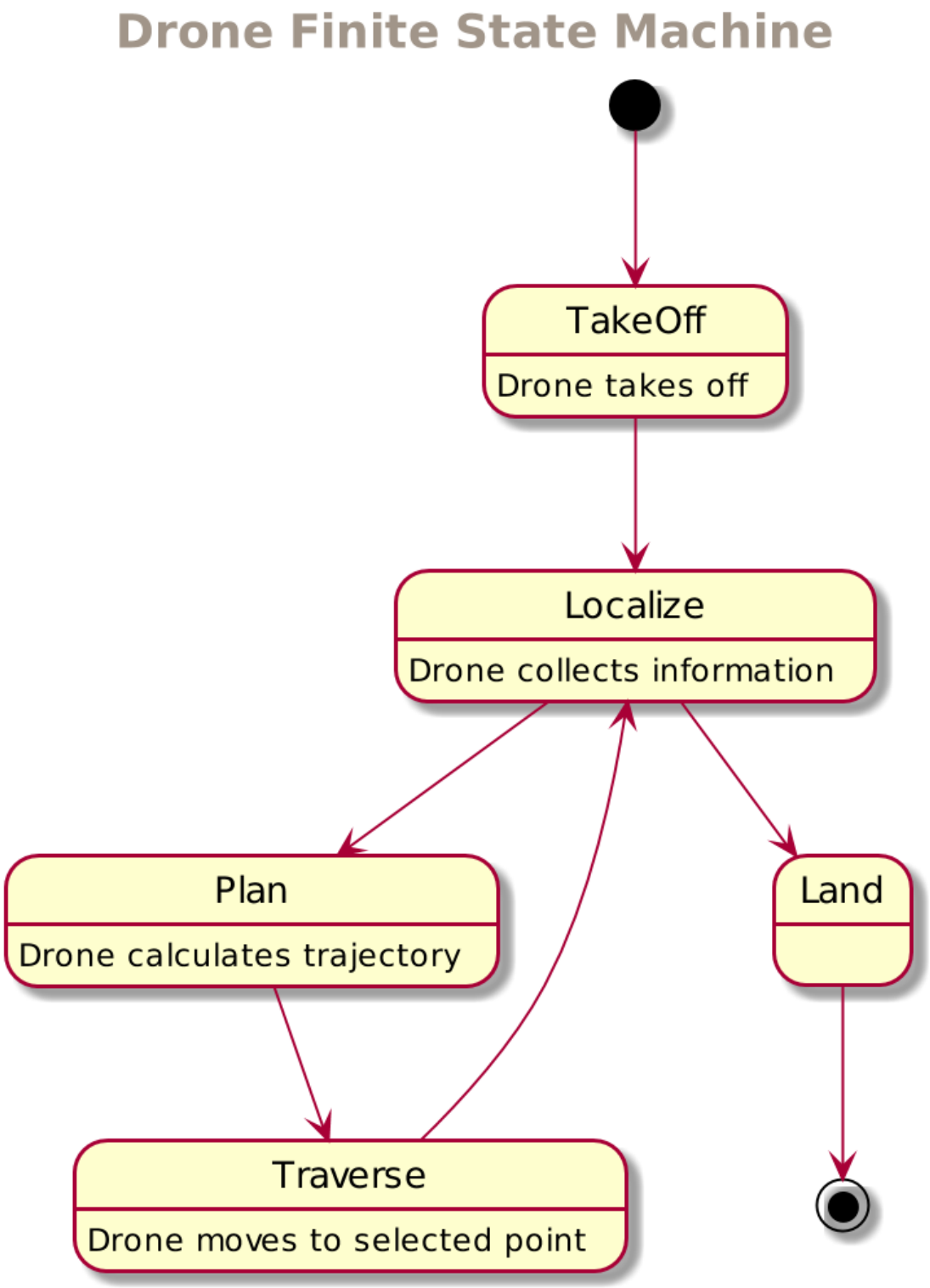
   ROS Toolbox allows you to access ROS functionality in MATLAB® . Use MATLAB to communicate with a ROS network, interactively explore robot capabilities, and visualize sensor data. You can develop robotics applications by exchanging data with ROS-enabled robots and robot simulators such as Gazebo. You can also create Simulink® models that exchange messages with a ROS network. Verify your model within the Simulink environment by receiving messages from, and sending messages to, ROS-enabled robots and robot simulators. From your model, you can also generate C++ code for a standalone ROS application.

   Both MATLAB and Simulink support the TCPROS transport layer (see TCPROS). The UDPROS transport is not supported.

   ROS Toolbox supports ROS Indigo and Hydro platforms, but your own ROS installation may have different message versions. If you would like to overwrite our current message catalog, you can utilize ROS Custom Message Support to generate new message definitions. For ROS 2, ROS Toolbox supports the Dashing Diademata platform.

3.3 Control Scheme

3.3.1 Drone Finite State Machine



So our drone needs a way to figure out what to do and how to do it. A really simple Finite State Machine is below. Should probably ask someone who knows what they're doing.

The SLAM algorithm is commputer intensive BUT once localization is done, it's pretty fast.

So our first pass will be super slow to collect info. Then, we can use the state machine to switch between

the Localize & Traverse states and make optimal use of information.

3.3.2 ORB-SLAM2

Status: Needs review

Github-Repo: https://github.com/raulmur/ORB_SLAM2
Youtube-Example: https://www.youtube.com/watch?v=IuBGKxgaxS0
ROS-Wiki Link: http://wiki.ros.org/orb_slam2_ros
License: GPLv3

ORB-SLAM2 is a real-time SLAM library for Monocular, Stereo and RGB-D cameras that computes the camera trajectory and a sparse 3D reconstruction (in the stereo and RGB-D case with true scale). It is able to detect loops and relocalize the camera in real time. We provide examples to run the SLAM system in the KITTI dataset as stereo or monocular, in the TUM dataset as RGB-D or monocular, and in the EuRoC dataset as stereo or monocular. We also provide a ROS node to process live monocular, stereo or RGB-D streams. The library can be compiled without ROS. ORB-SLAM2 provides a GUI to change between a SLAM Mode and Localization Mode, see section 9 of this document.

1. ROS Parameters

   There are three types of parameters right now: static- and dynamic ros parameters and camera settings from the config file. The static parameters are send to the ROS parameter server at startup and are not supposed to change. They are set in the launch files which are located at ros/launch. The parameters are:

   - `load_map`: Bool. If set to true, the node will try to load the map provided with map_{file} at startup.
   - `map_file`: String. The name of the file the map is saved at.
   - `settings_file`: String. The location of config file mentioned above.
   - `voc_file`: String. The location of config vocanulary file mentioned above.
   - `publish_pose`: Bool. If a PoseStamped message should be published. Even if this is false the tf will still be published.
   - `publish_pointcloud`: Bool. If the pointcloud containing all key points (the map) should be published.
   - `pointcloud_frame_id`: String. The Frame id of the Pointcloud/map.
   - `camera_frame_id`: String. The Frame id of the camera position.

   Dynamic parameters can be changed at runtime. Either by updating them directly via the command line or by using $\text{rqt}_{\text{reconfigure}}$ which is the recommended way. The parameters are:

   - `localize_only`: Bool. Toggle from/to only localization. The SLAM will then no longer add no new points to the map.
   - `reset_map`: Bool. Set to true to erase the map and start new. After reset the parameter will automatically update back to false.
   - `min_num_kf_in_map`: Int. Number of key frames a map has to have to not get reset after tracking is lost.

   Finally, the intrinsic camera calibration parameters along with some hyperparameters can be found in the specific yaml files in $\text{orb}_{\text{slam2}}$/config.

2. ROS Subscribed topics

   - The `mono` node subscribes to `/camera/image_raw` for the input image.
   - The RGBD node subscribes to `/camera/rgb/image_raw` for the RGB image and
   - `/camera/depth_registered/image_raw` for the depth information.
   - The stereo node subscribes to `image_left/image_color_rect` and
   - `image_right/image_color_rect` for corresponding images.

3. ROS Published topics

   The following topics are being published and subscribed to by the nodes:

   - All nodes publish (given the settings) a PointCloud2 containing all key points of the map.
   - Live image from the camera containing the currently found key points and a status text.
   - A tf from the pointcloud frame id to the camera frame id (the position).

4. ROS Services

   All nodes offer the possibility to save the map via the service $node_{type}/save_{map}$. So the $save_{map}$ services are:

   - `/orb_slam2_rgbd/save_map`
   - `/orb_slam2_mono/save_map`
   - `/orb_slam2_stereo/save_map`

3.3.3 Unscented Kalman Filter

Status: Needs review

Like an Extended Kalman Filter but more performant. Able to deal with the drone's non-linearities and should give us a decent idea of where and how fast our drone is moving.

for the accelerometer, gyroscope and stuff This is what keeps the drone actually flying in the air. The ORB-SLAMv2 is really just a way to identify points

4 TODO Action items

4.1 TODO Base Simulink Simulation

Also need to look at the base model (and add the existing stuff to git repo)

Some hard numbers instead of qualitative stuff.

4.1.1 TODO Make comparison of models

- ROS vs Simulink
- ORB-SLAM vs other vSLAM models
- Different Feature Detection inside vSLAM

4.1.2 TODO Simulink Model in Gitlab Repo

Add the base case to the gitlab repo for convenience.

Notify Vishnu when done.

4.1.3 Debugging

Useful link: https://www.mathworks.com/matlabcentral/answers/?term=parrot

4.2 TODO ORB-SLAM Simulink model

Implement ORB-SLAM in Simulink (again, ughh).

4.3 TODO Need to get real-life data for stuff

Need to get Dr Mehdi to send over a recording of sensor data from the drone. While we can't actually work with said drone thanks to the whole lockdown, having some raw data should allow us to make (somewhat rational) decisions about which filters and what parameters to use to analyze said data.

Not a priority (yet) and I think Dr Mehdi is busy with exam stuff. Would recommend bugging him about it before our meeting on May 17.

4.4 TODO Equations of motion

For the UKF because you need a proper statespace model

4.4.1 Need some controlability analysis

Mostly so that we know we're not missing out on some really useful info.

Give Sheriff some equations to use the SVD analysis on Two birds with one stone - Controllability analysis as well as practice for Sheriff.