Core Autonomy Stack

Building

- Setting up the code and building

  https://bitbucket.org/castacks/core_central/src/master/

  ```
  mkdir -p ws/src
  cd ws/src
  git clone git@bitbucket.org:castacks/core_central.git
  ln -s core_central/rosinstalls/sim.rosinstall .rosinstall
  wstool up -j 16
  cd ..
  catkin build
  ```

- Updating

  ```
  cd ws/src/core_central
  git pull
  cd ..
  wstool up -j 16
  catkin build
  ```

Core Autonomy Stack

- Set of packages for basic autonomy
  - Takeoff/Landing
  - Obstacle Avoidance
  - Base Station GUI

Motivation

- Basic software stack for getting a drone flying autonomously
- Starting point for new projects
- Avoid every project having their own setup
- Easy to use
- Easy to replace components

History

- Autel project
  - Camera only drone
  - A lot of the code was in a single node
- Subt and Autel
  - Subt drone has a LIDAR
  - local_planner, trajectory_controller, etc...
- Core Stack
  - core_local_planner, core_trajectory_controller, etc..

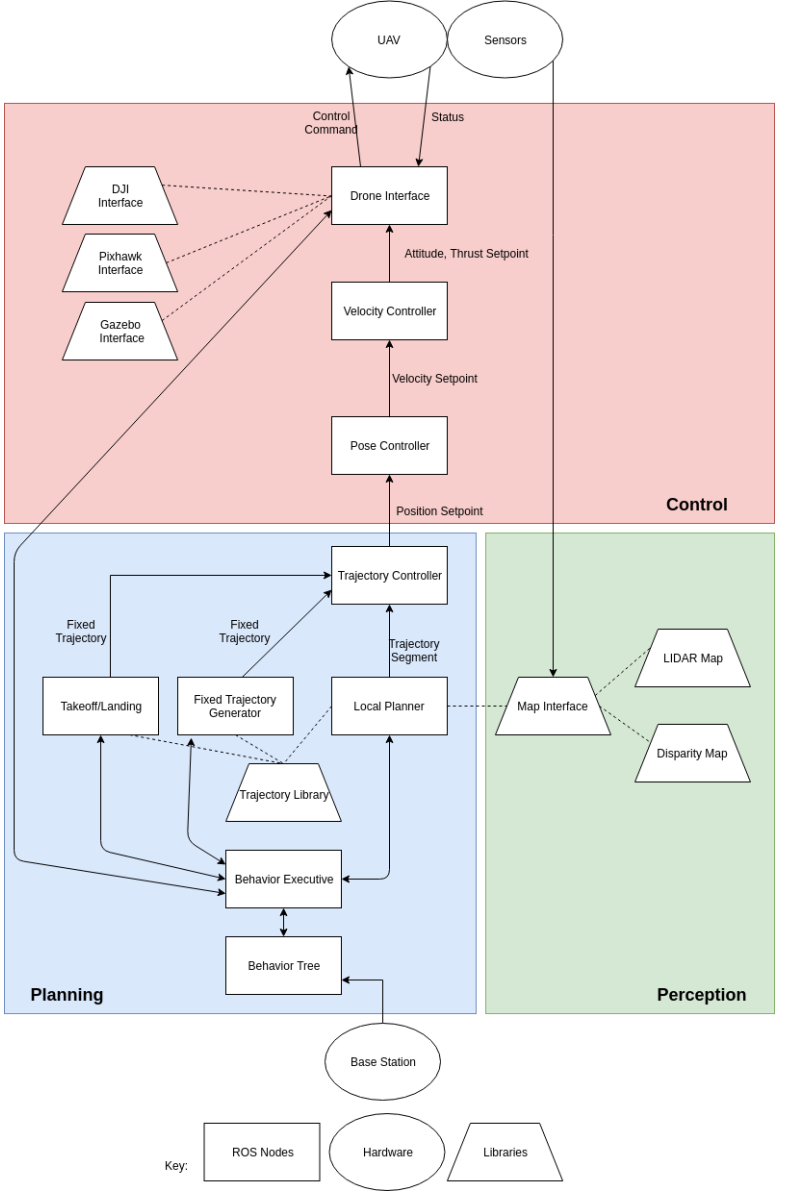
Drones Using The Stack

- Subt hex and quad
- MBZ M210

Outline

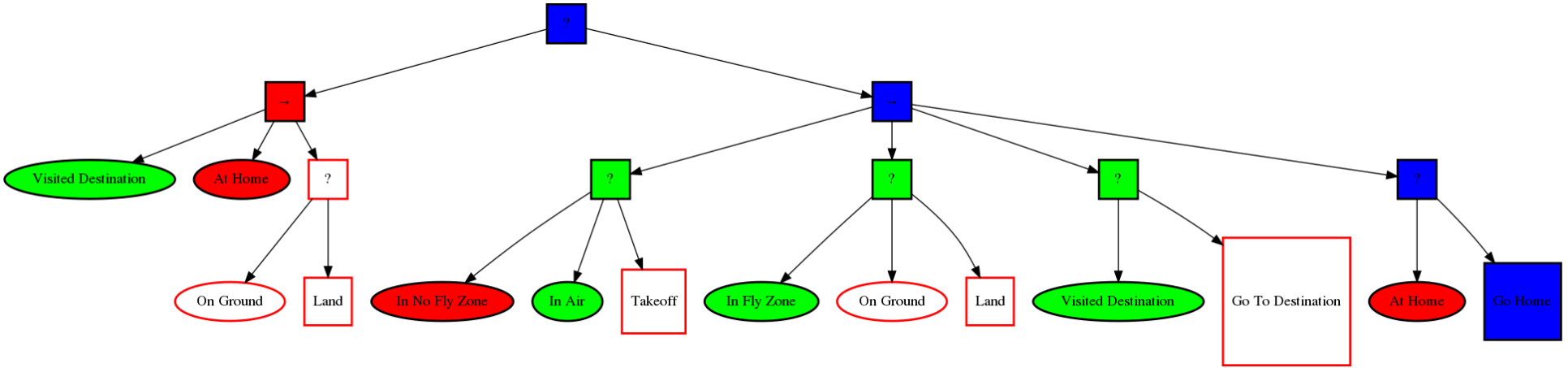
- Overview of software architecture
- How to use the stack
- Exercise

Architecture



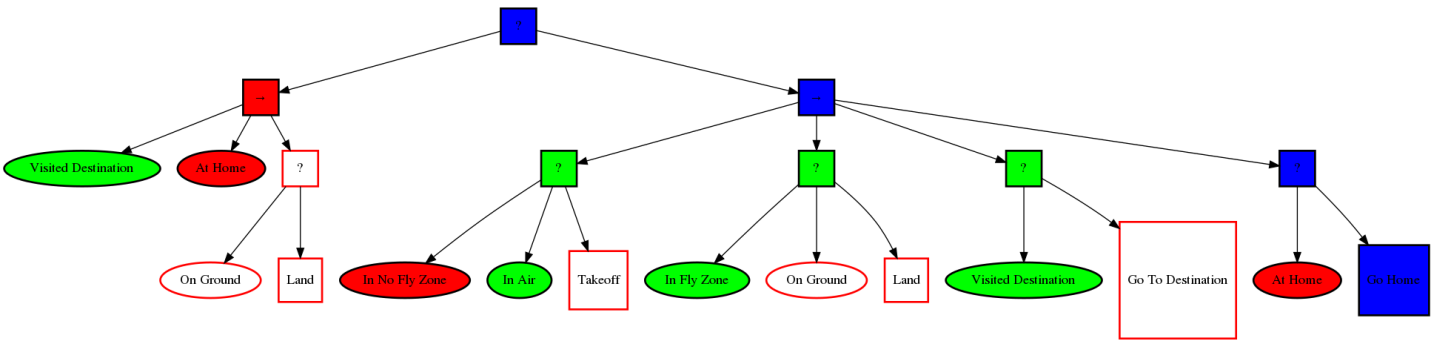
Behavior Tree

- Chooses which task to perform
- Condition and Action nodes
- Control Flow Nodes
  - Sequence →, Fallback ?



Behavior Tree

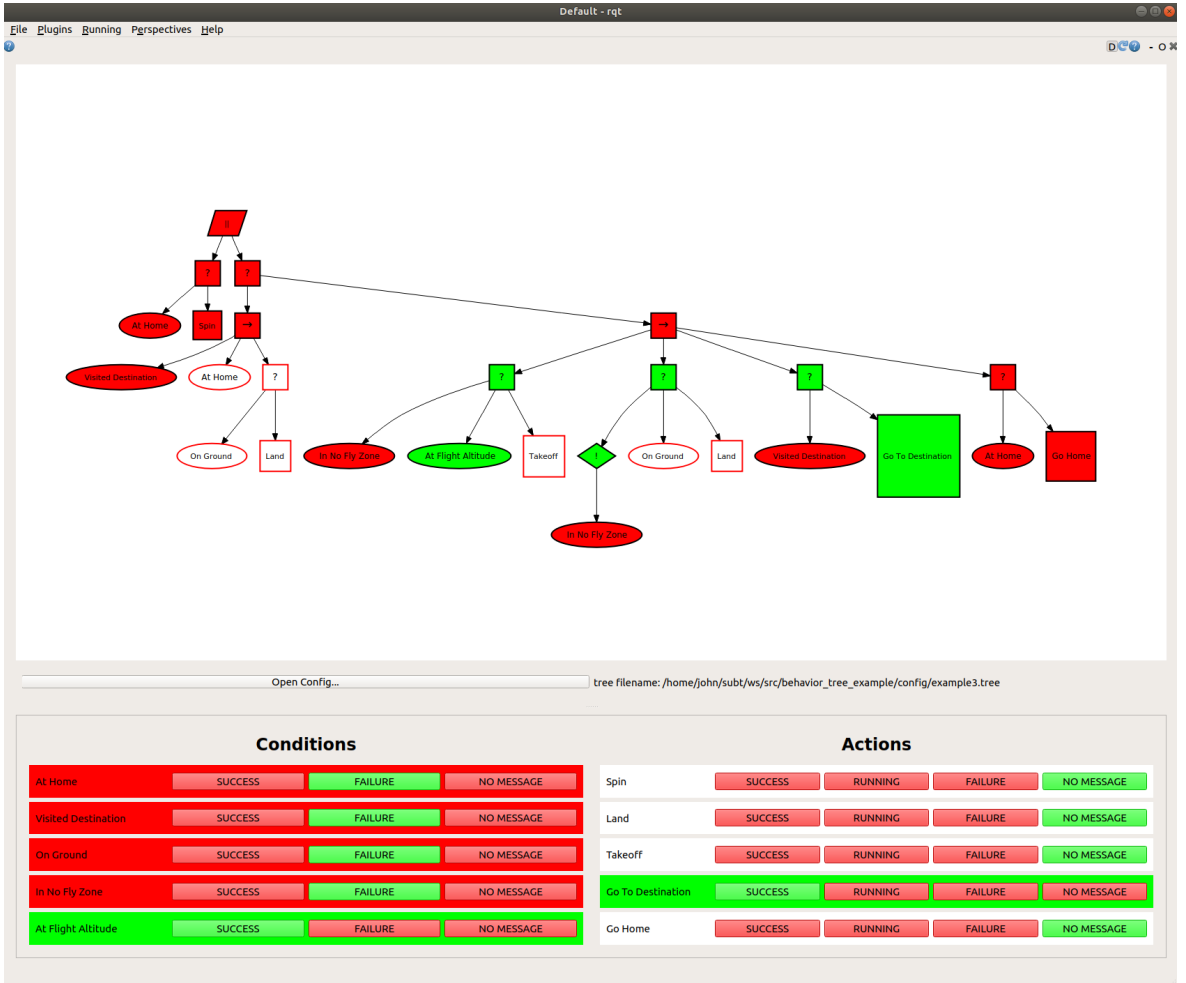
- Config file makes it easy to modify the behavior tree



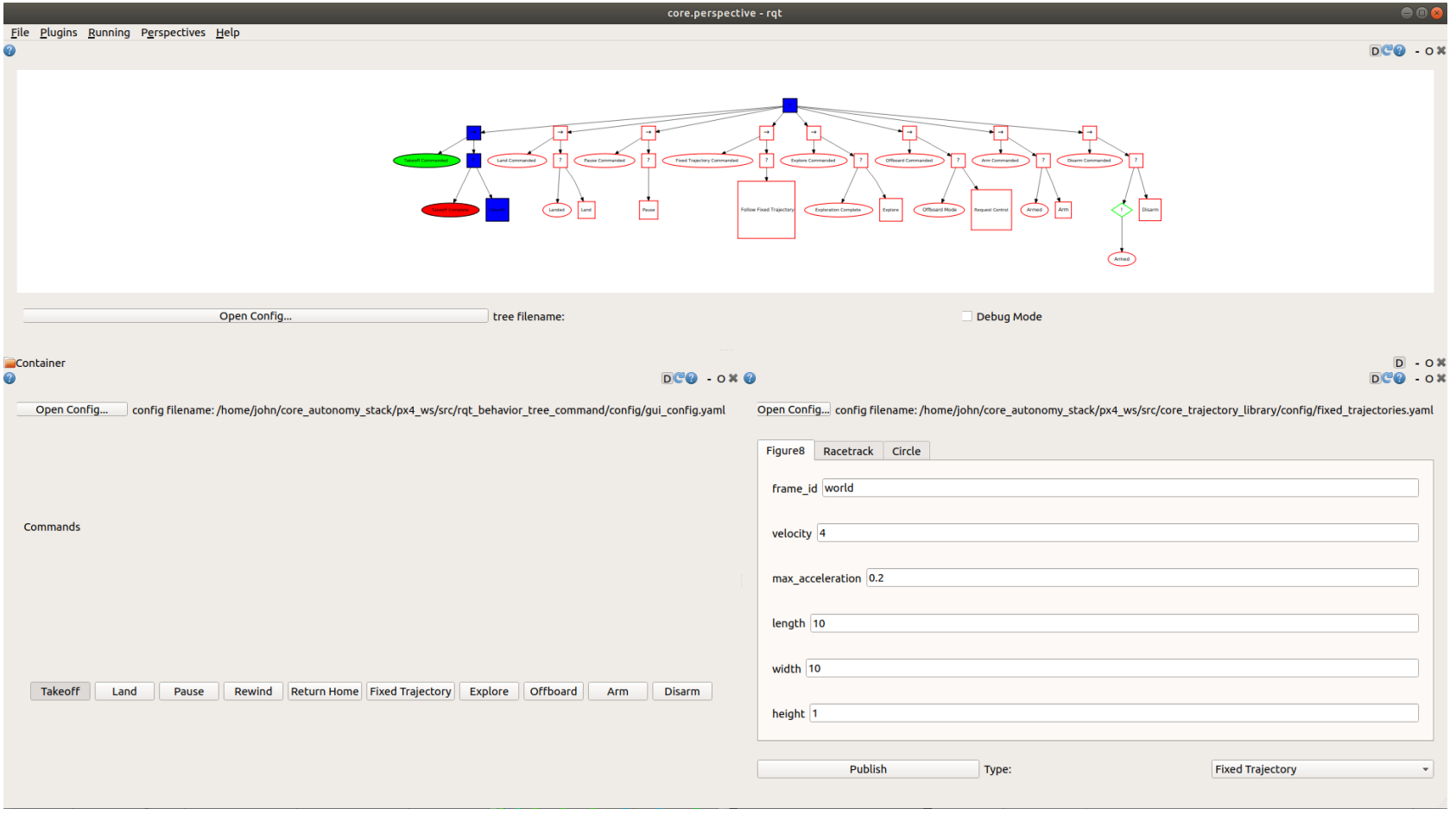

```
1     ?
2         ->
3             (Visited Destination)
4             (At Home)
5             ?
6                 (On Ground)
7                 [Land]
8         ->
9             ?
10                (In No Fly Zone)
11                (At Flight Altitude)
12                [Takeoff]
13            ?
14                (In Fly Zone)
15                (On Ground)
16                [Land]
17            ?
18                (Visited Destination)
19                [Go To Destination]
20            ?
21                (At Home)
22                [Go Home]
23
```

Behavior Tree

- GUI for debugging the structure of a tree

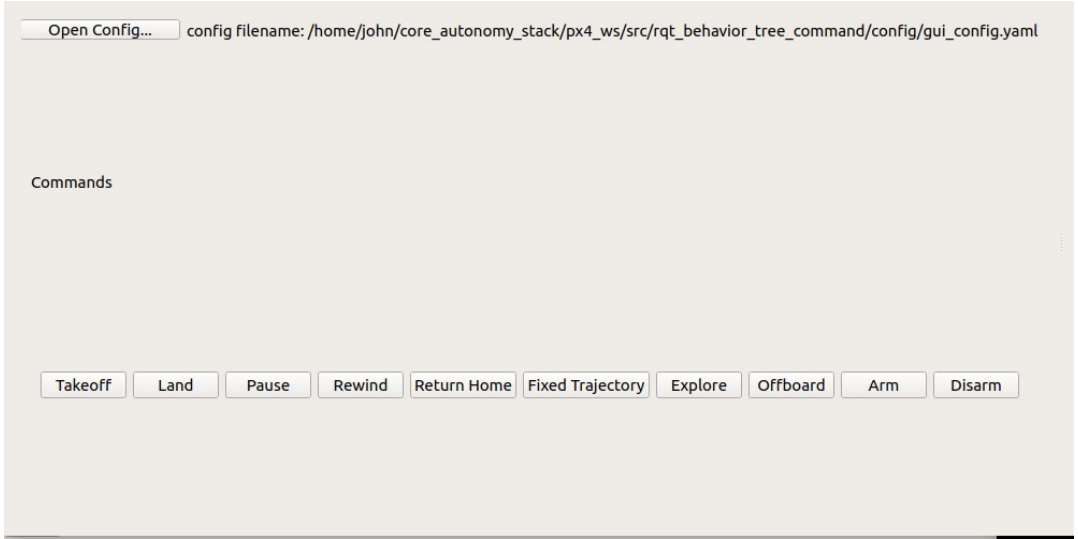


Base Station GUI



Base Station GUI

- Only one button within each group can be pressed, the rest become unpressed. There is currently only one group, “Commands”.
- Each button controls the value of a condition in the behavior tree

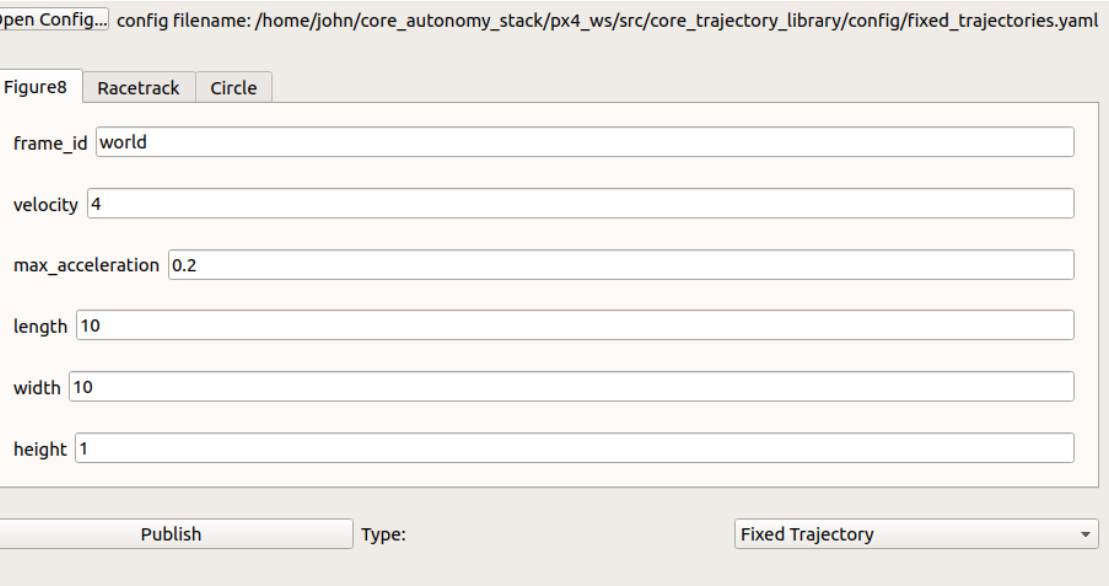



```yaml
groups:
  - Commands:
      - Takeoff:
          condition_name: Takeoff Commanded
      - Land:
          condition_name: Land Commanded
      - Pause:
          condition_name: Pause Commanded
      - Rewind:
          condition_name: Rewind Commanded
      - Return Home:
          condition_name: Return Home Commanded
      - Fixed Trajectory:
          condition_name: Fixed Trajectory Commanded
      - Explore:
          condition_name: Explore Commanded
      - Offboard:
          condition_name: Offboard Commanded
      - Arm:
          condition_name: Arm Commanded
      - Disarm:
          condition_name: Disarm Commanded
```

Base Station GUI

- The config yaml file defines trajectories and their attributes. Trajectories are published to a message containing the attributes as key, value pairs, which the fixed trajectory geneator subscribes to and publishes a trajectory containing waypoints the drone can follow.

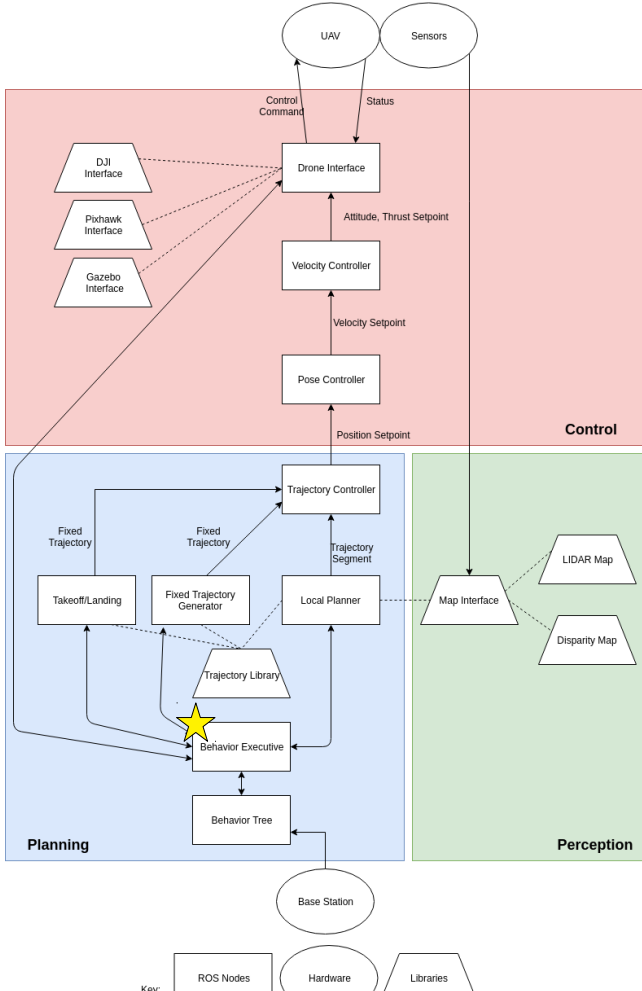
- The Type menu determines whether the trajectory will be a “Fixed Trajectory” that the robot follows blindly for control tuning, or a “Global Plan” that the local planner will try to follow while avoiding obstacles.




```yaml
trajectories:
  - Figure8:
      attributes:
        - frame_id
        - velocity
        - max_acceleration
        - length
        - width
        - height
  - Racetrack:
      attributes:
        - frame_id
        - velocity
        - max_acceleration
        - length
        - width
        - height
  - Circle:
      attributes:
        - frame_id
        - velocity
        - radius
```

Behavior Executive

- Reports statuses of action and conditions to the behavior tree
- Implements what happens when an action is active



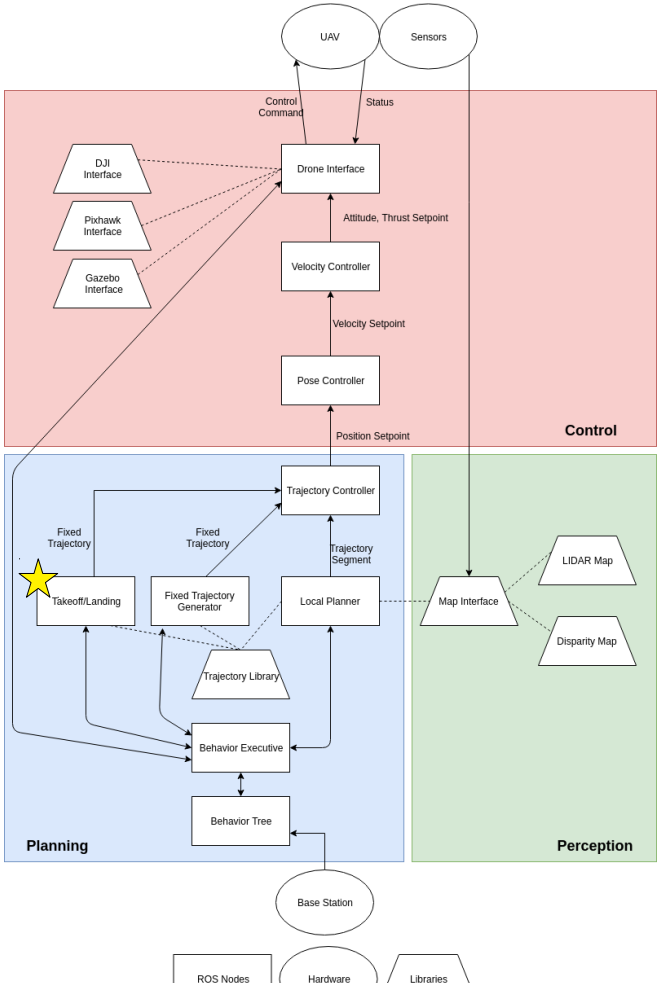
Behavior Executive

- Example Takeoff action

```
// Initialize an action with the same name it has in the behavior tree config file.
takeoff_action = new bt::Action("Takeoff");

// Takeoff Action
if(takeoff_action->is_active()){
  // Tell the behavior tree that the action is running.
  takeoff_action->set_running();

  // When the action first becomes active, make a service call to the takeoff landing planner.
  if(takeoff_action->active_has_changed()){
    core_takeoff_landing_planner::TakeoffLandingCommand takeoff_srv;
    takeoff_srv.request.command = core_takeoff_landing_planner::TakeoffLandingCommand::Request::TAKEOFF;
    takeoff_landing_client.call(takeoff_srv);
  }

  // When the takeoff landing planner tells us the takeoff is complete, tell the behavior tree the action has succeeded.
  if(takeoff_state == "COMPLETE"){
    takeoff_complete_condition->set(true);
    takeoff_action->set_success();
  }

  // publish the status
  takeoff_action->publish();
}
```

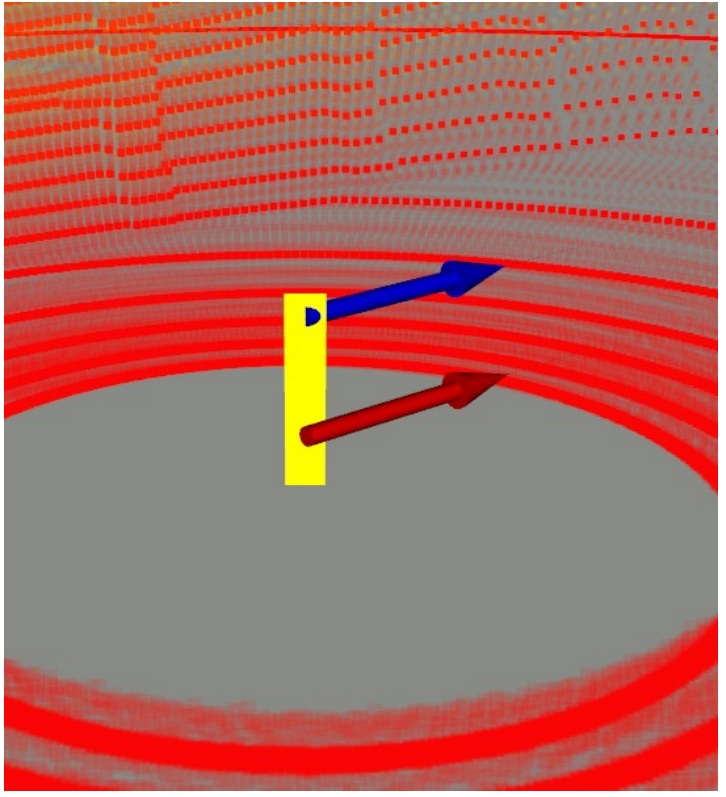
Takeoff Landing Planner

- Publishes fixed vertical takeoff and landing trajectories and monitors when the drone completes them



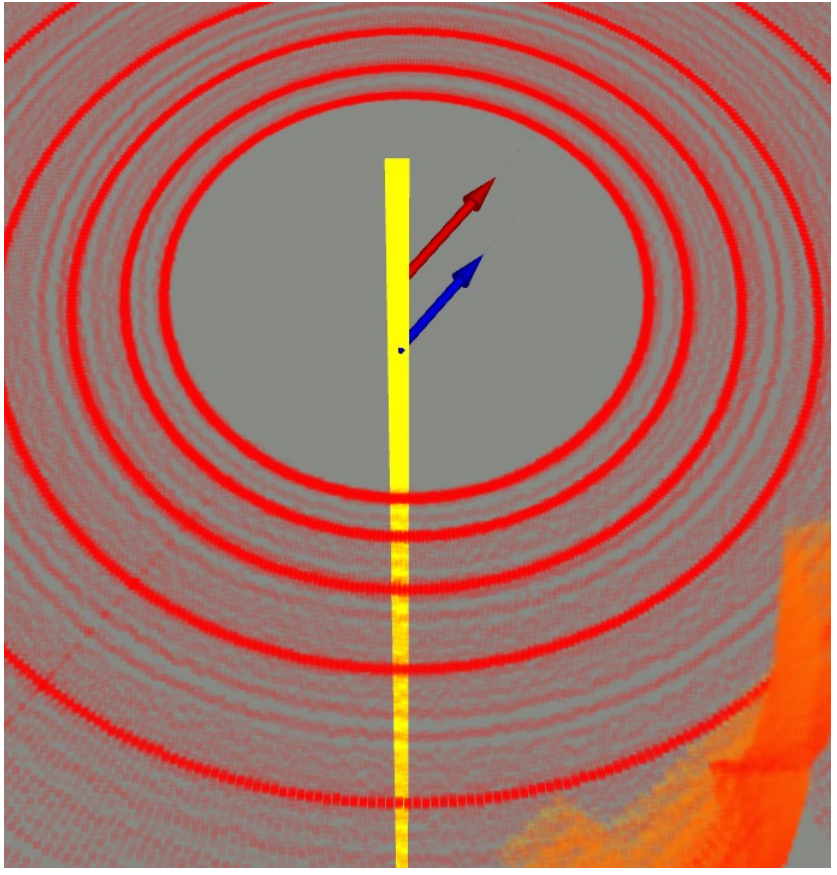
Takeoff Landing Planner

- Takeoff Example
  - Tracking Point, Robot's Pose
  - Complete when tracking point is at top of trajectory, and robot's pose has been within some threshold distance for some threshold time



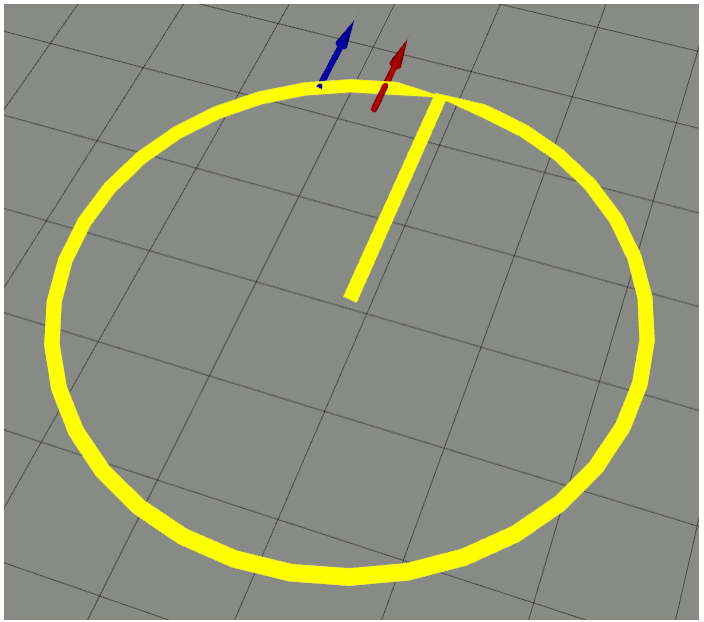
Takeoff Landing Planner

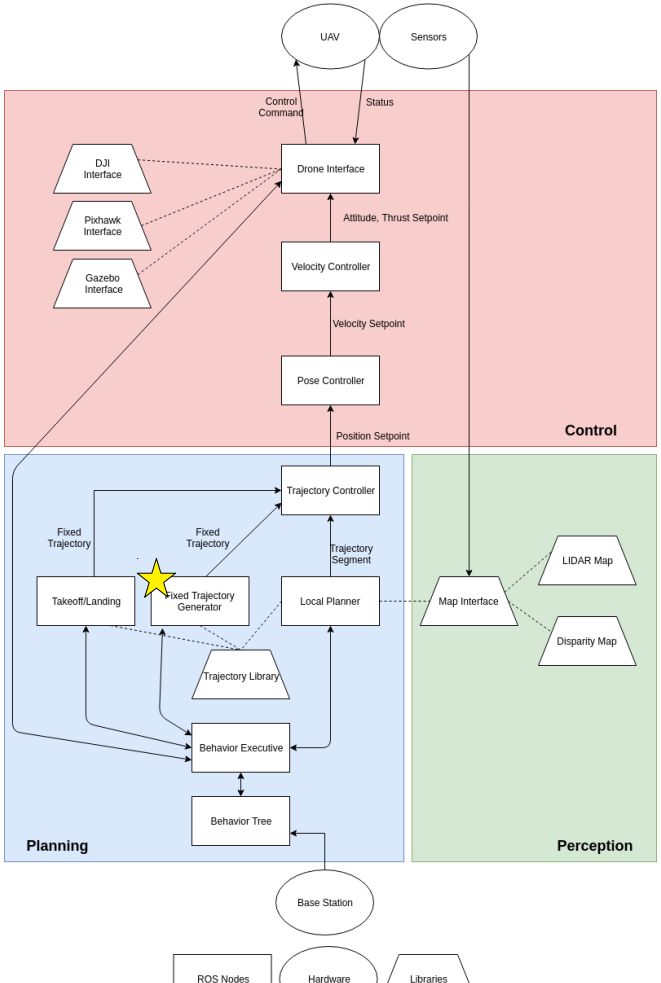
- Landing Example
  - Publishes very long downward trajectory
  - Complete when the robot’s position hasn’t moved more than some threshold distance for some threshold time



Fixed Trajectory Generator

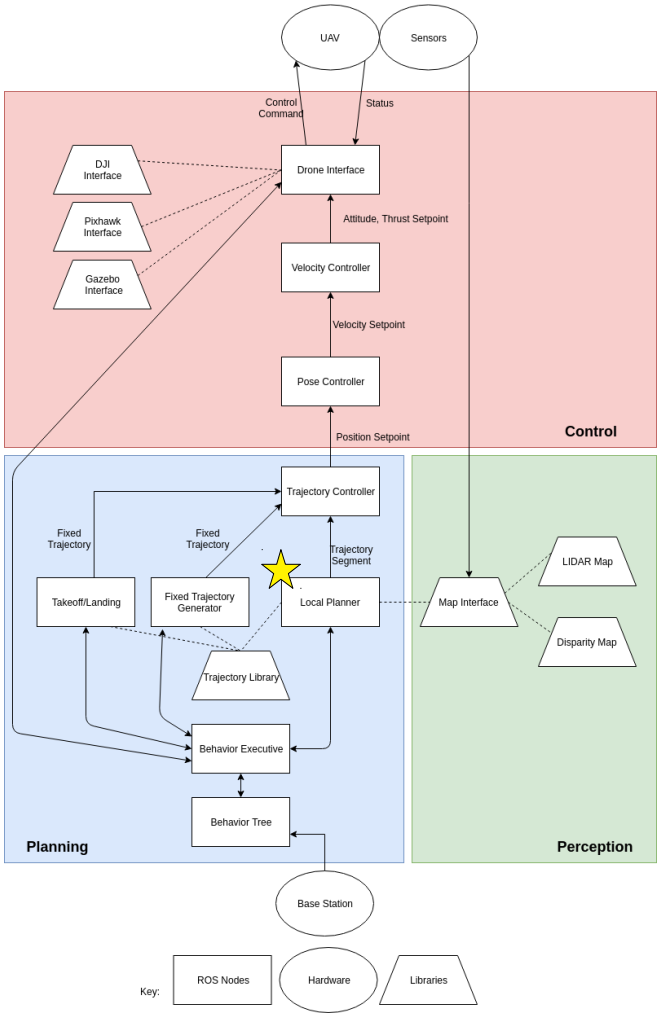
- Publishes circle, racetrack, and figure eight trajectory, mainly for tuning control gains





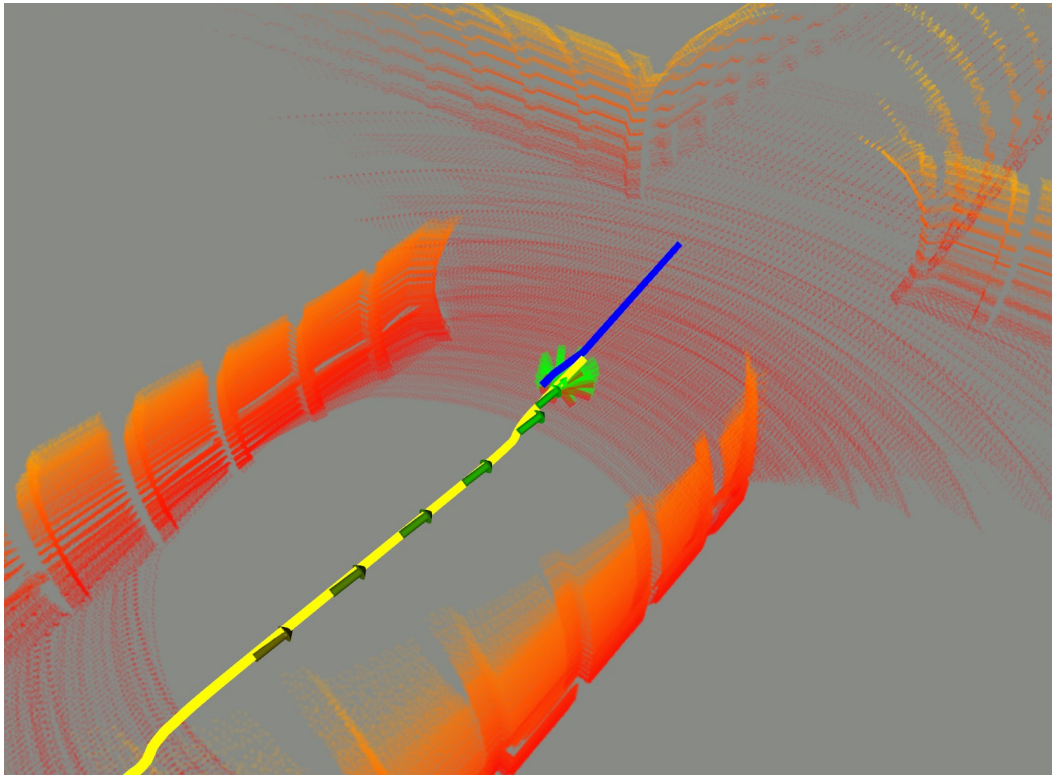
Local Planner

- Chooses a trajectory from a trajectory library to avoid obstacles and follow a global plan



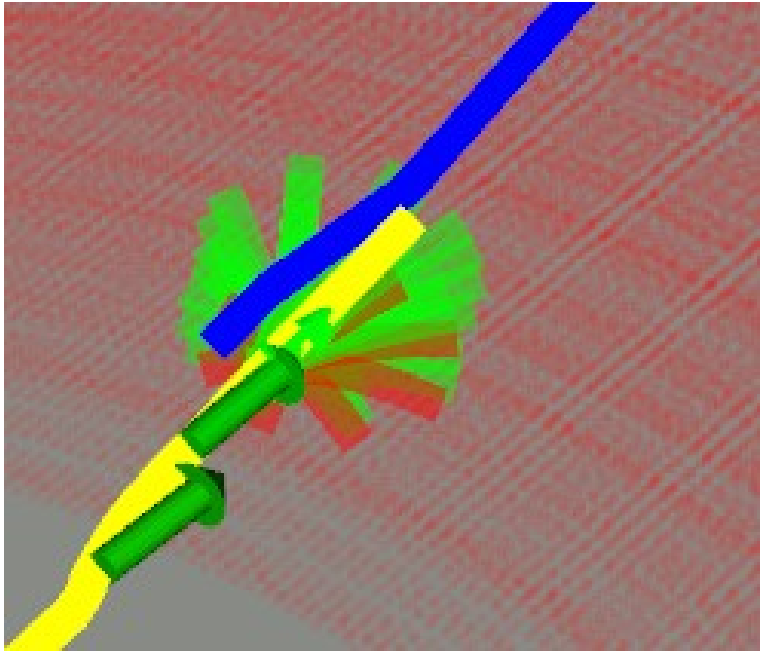
Local Planner

- Chooses a trajectory from a trajectory library to avoid obstacles and follow a global plan



Trajectory Library

- Set of trajectories defined by accelerations



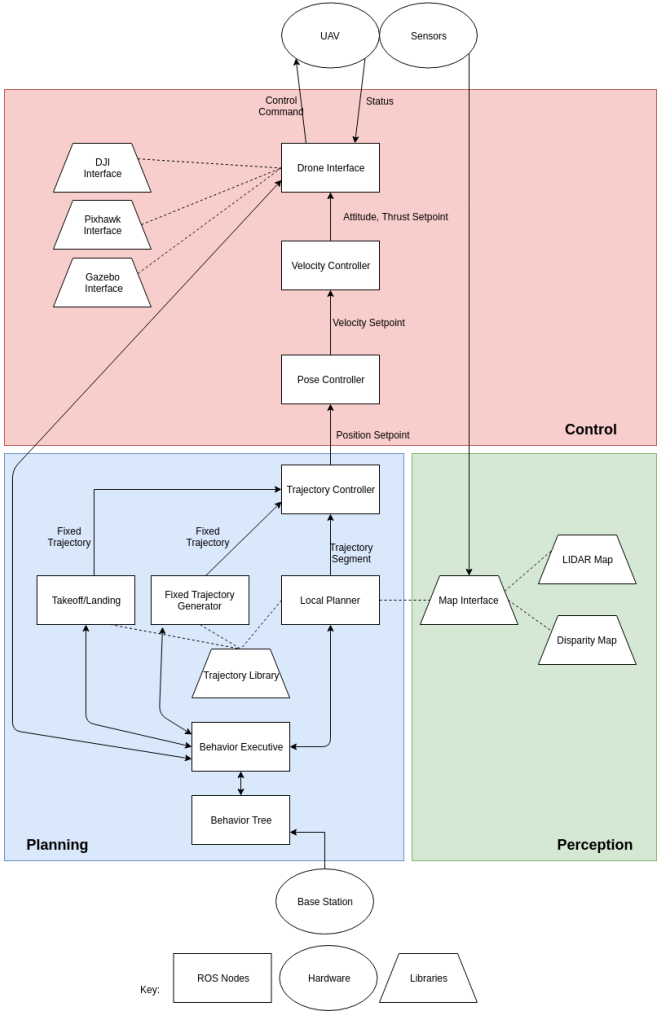
Trajectory Library

- Example yaml config file
  - Two trajectories
  - The $(param) will get its value from roslaunch parameters

```yaml
trajectories:
  - type: acceleration
    frame: $(param tf_prefix)/look_ahead_point_stabilized
    magnitude: $(param magnitude)
    magnitude_yaw: 0.
    magnitude_pitch: 0.
    dt: $(param dt)
    ht: $(param ht)
    max_velocity: $(param max_velocity)
  - type: acceleration
    frame: $(param tf_prefix)/look_ahead_point_stabilized
    magnitude: $(param magnitude)
    magnitude_yaw: 45.
    magnitude_pitch: 0.
    dt: $(param dt)
    ht: $(param ht)
    max_velocity: $(param max_velocity)
```

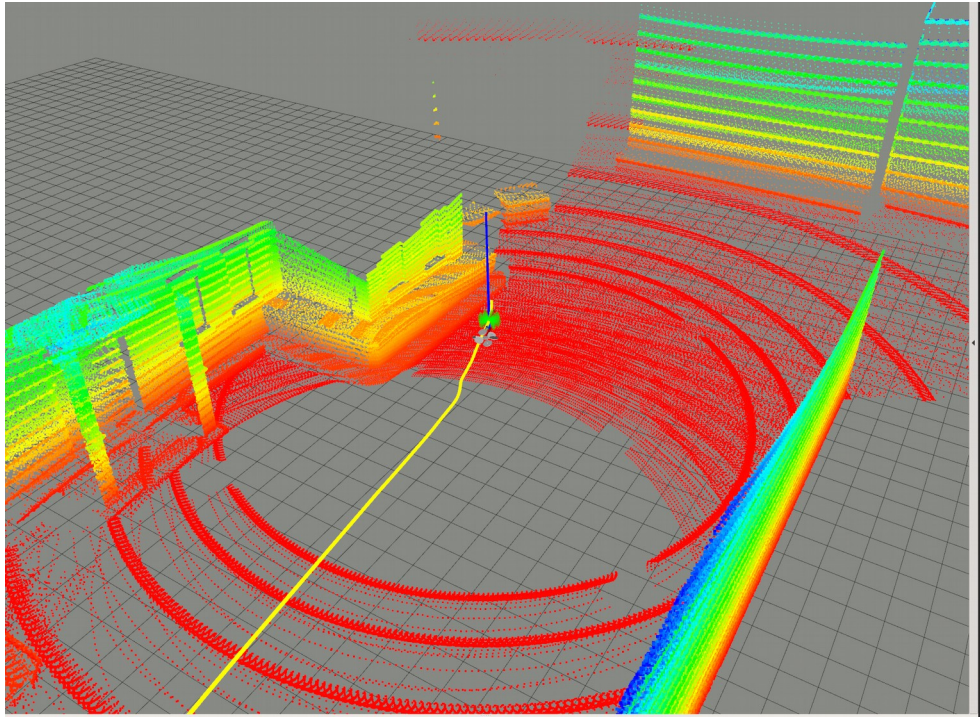
Map Representation

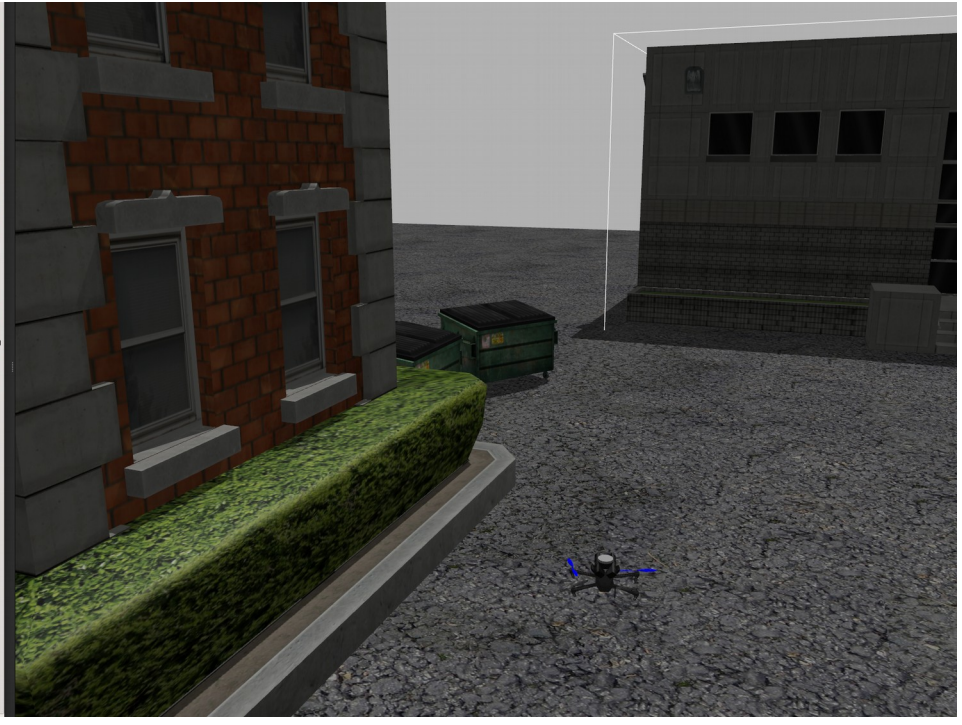
- Disparity and LIDAR representations
- Uses PluginLib to easily choose which map representation library to load in at runtime with a launch file parameter.
- Used by the local planner for collision checking



Map Representation

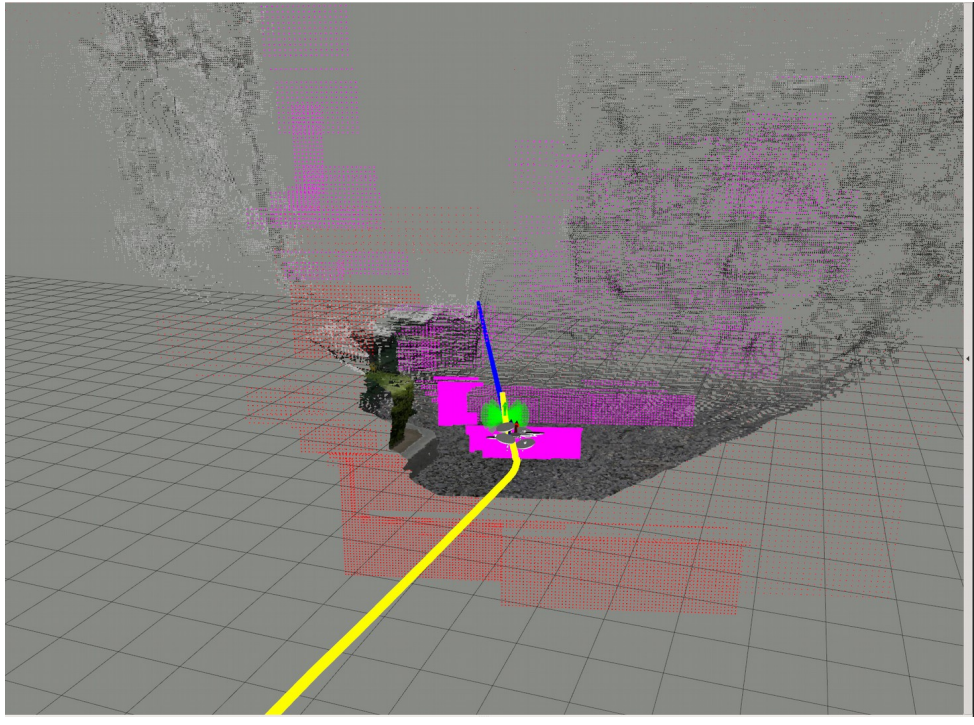
- LIDAR based map, pose graph of kd-trees





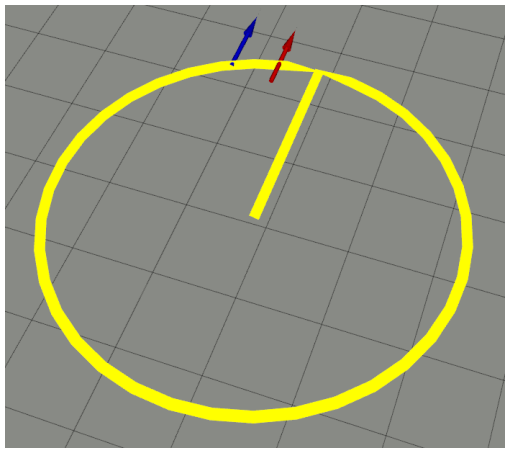
Map Representation

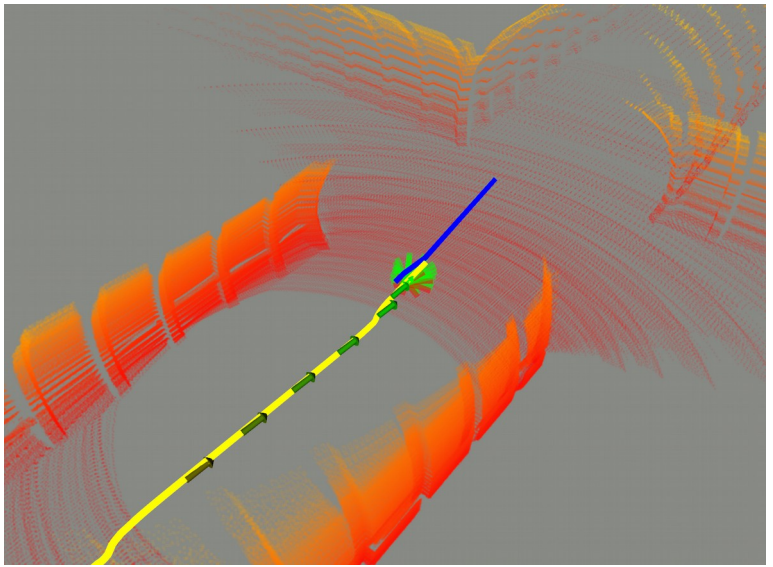
- Disparity based map, pose graph of expanded disparity images

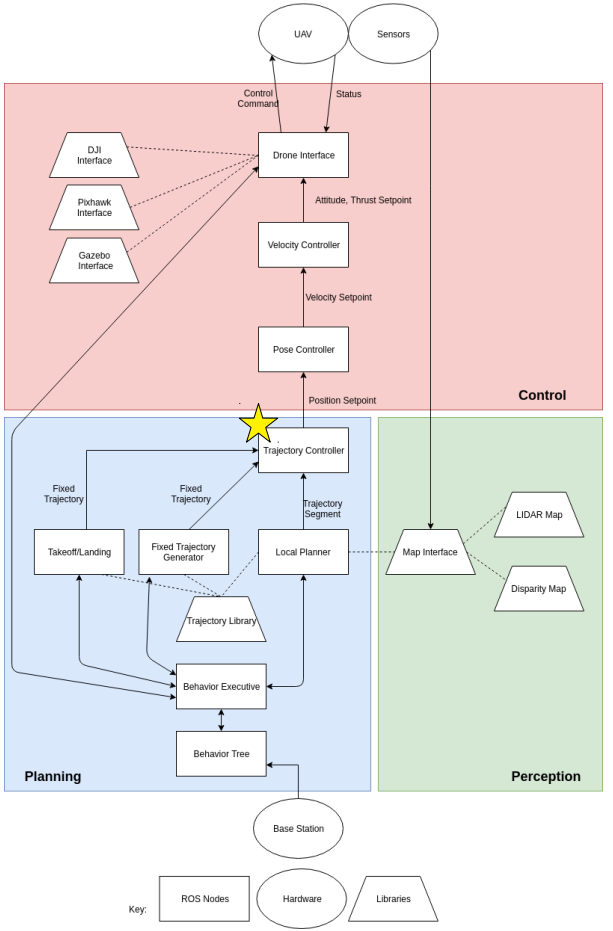


Trajectory Controller

- Takes in complete trajectories or trajectory segments to stitch together

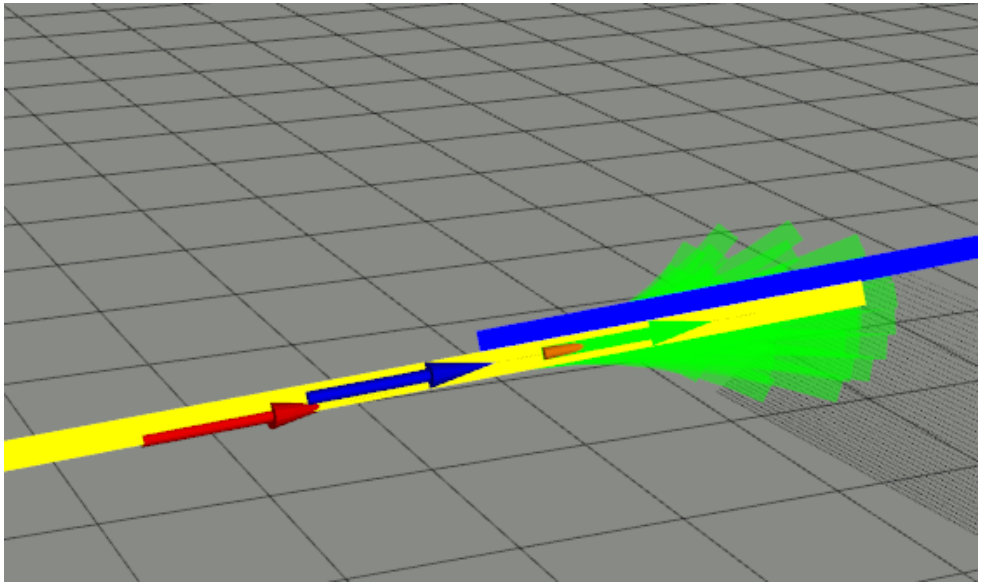


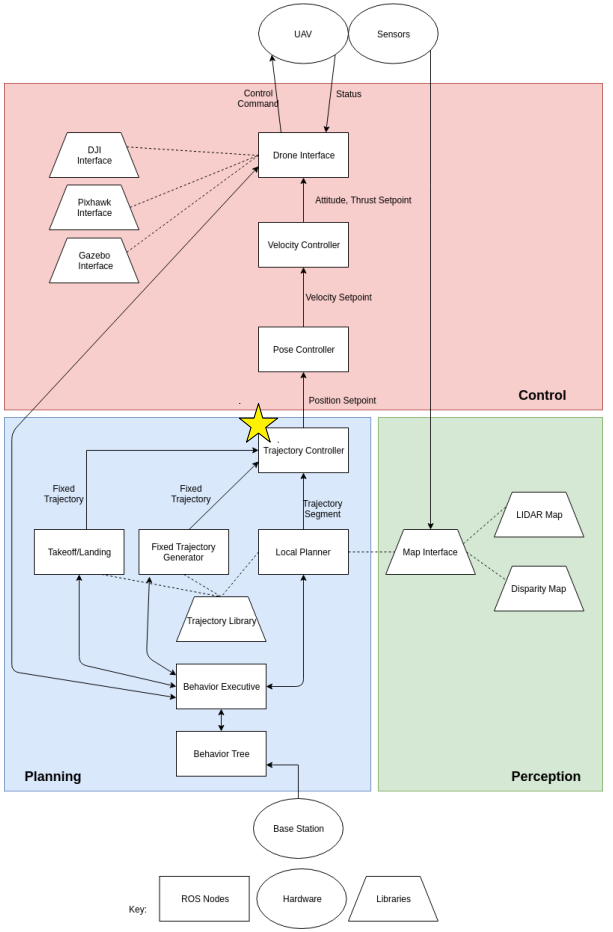




Trajectory Controller

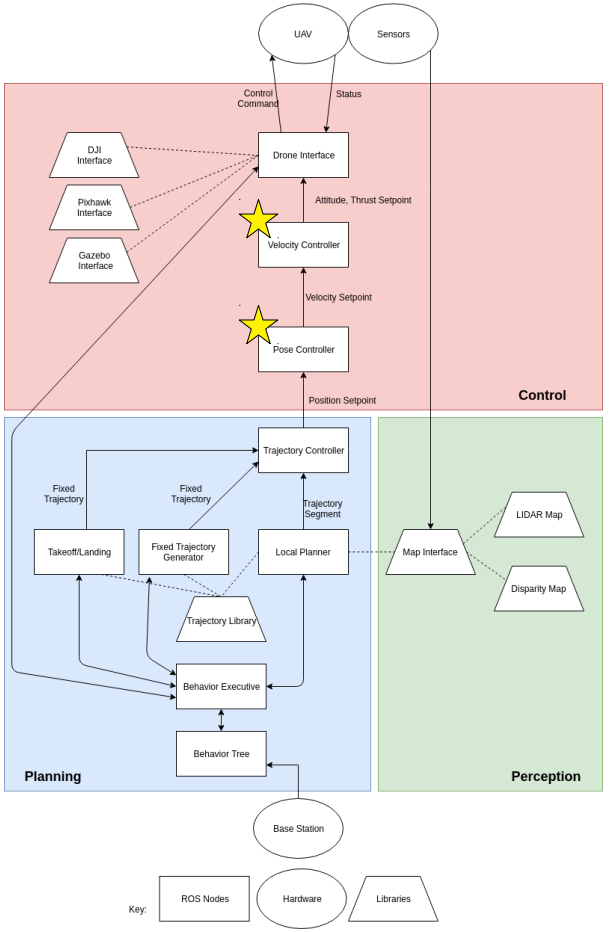
- Outputs a tracking point for the controllers to follow and look ahead point to plan from
- Can also pause and rewind the tracking point





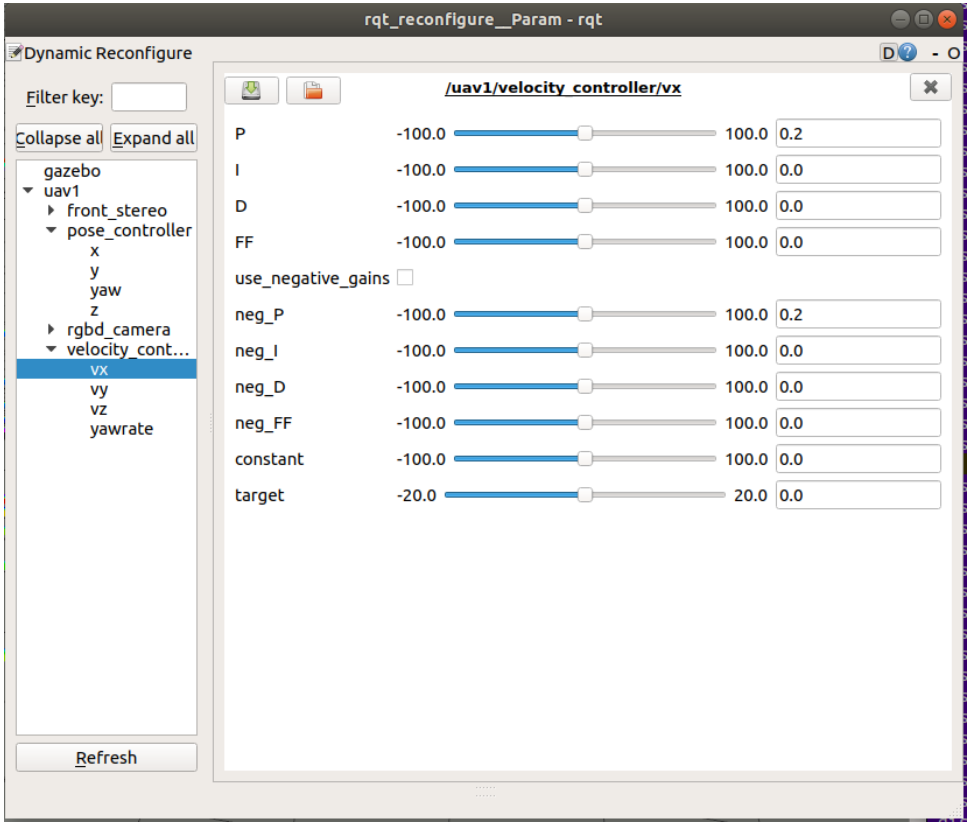
Pose and Velocity Controllers

- Both use a common PID controller class
- Pose controller has x, y, z, and yaw PID controller instances
  - Takes in the tracking point Odometry message from the trajectory controller
  - Outputs a velocity setpoint with the velocity from the Odometry message as a feed forward term plus a PID controller on the robot's position error
- Velocity controller has vx, vy, vz, and yawrate PID controller instances
  - Takes in a velocity setpoint
  - Outputs a roll, pitch, yawrate, thrust command



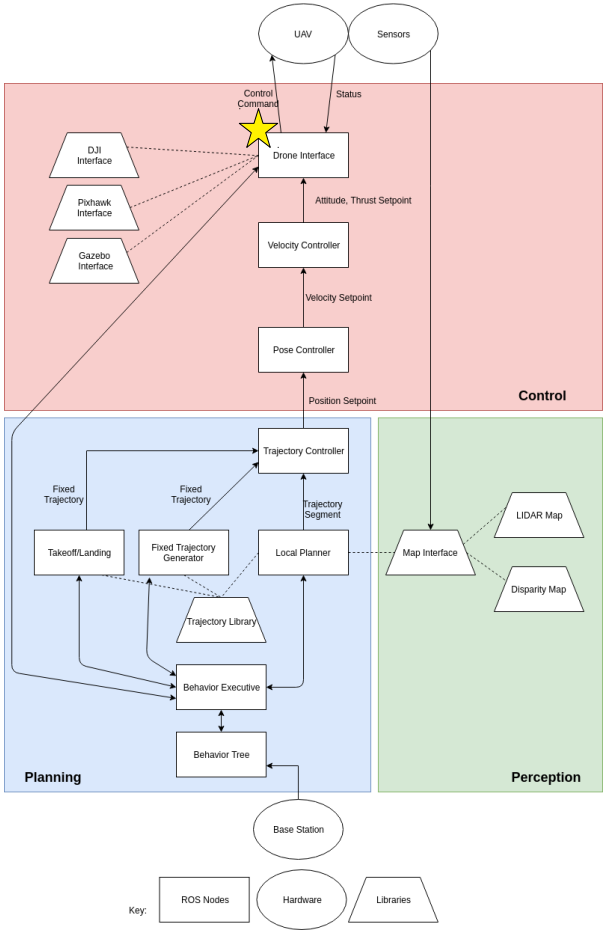
Pose and Velocity Controllers

- PID class uses dynamic reconfigure
- Mainly useful for sim
  - Not recommended for a real robot during flight



Drone Interface

- Provides a common interface for interacting with different platforms
- Currently implemented interfaces
  - Gazebo
  - PX4
  - DJI
- Uses PluginLib



Drone Interface

- request_control, arm, disarm, is_armed, and has_control must be implemented
- Whichever control command functions are relevant to a specific platform can be implemented

```
virtual bool request_control() = 0;
virtual bool arm() = 0;
virtual bool disarm() = 0;
virtual bool is_armed() = 0;
virtual bool has_control() = 0;

// command functions
virtual void command_attitude_thrust(mav_msgs::AttitudeThrust){
  ROS_ERROR("command_attitude_thrust WAS CALLED, BUT IS NOT IMPLEMENTED.");
}

virtual void command_rate_thrust(mav_msgs::RateThrust){
  ROS_ERROR("command_rate_thrust WAS CALLED, BUT IS NOT IMPLEMENTED.");
}

virtual void command_roll_pitch_yawrate_thrust(mav_msgs::RollPitchYawrateThrust){
  ROS_ERROR("command_roll_pitch_yawrate_thrust WAS CALLED, BUT IS NOT IMPLEMENTED.");
}

virtual void command_torque_thrust(mav_msgs::TorqueThrust){
  ROS_ERROR("command_torque_thrust WAS CALLED, BUT IS NOT IMPLEMENTED.");
}

virtual void command_velocity(geometry_msgs::TwistStamped){
  ROS_ERROR("command_velocity WAS CALLED, BUT IS NOT IMPLEMENTED.");
}

virtual void command_pose(geometry_msgs::PoseStamped){
  ROS_ERROR("command_position WAS CALLED, BUT IS NOT IMPLEMENTED.");
}
```

State Estimate

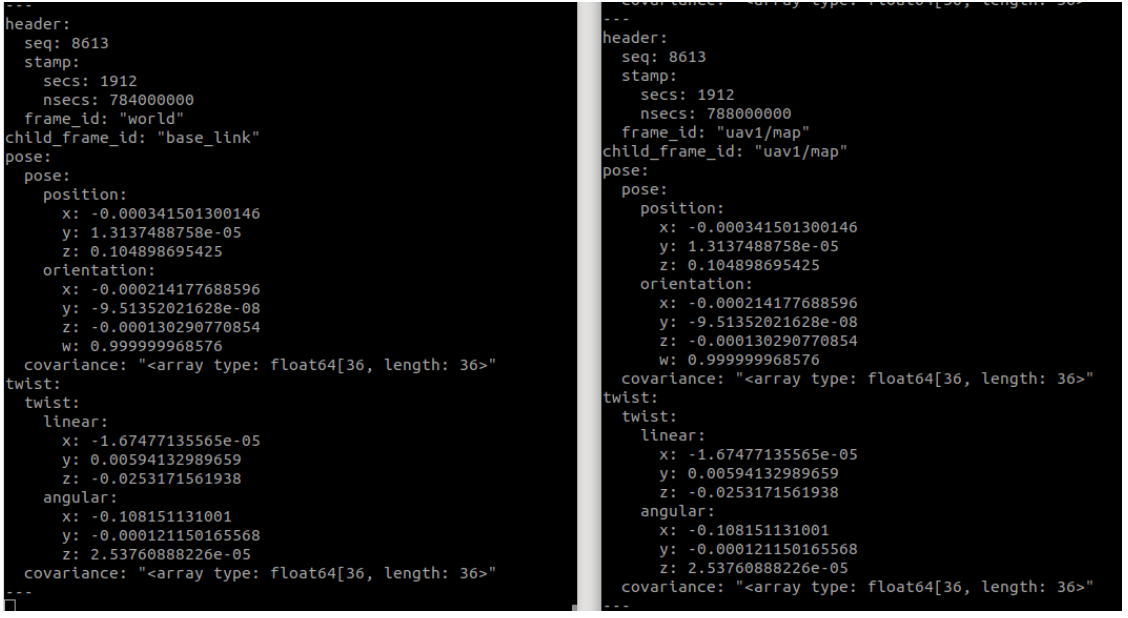
- Most packages in the stack assume there is a state estimate as a nav_msgs/Odometry message published on the uav#/odometry topic
- The frame_id and child_frame_id must be correct and be tfs that are in the tf tree
- core_odometry_transform
  - Package for helping with state estimate related odometry messages and transforms

Core Odometry Transform

- Can transform an input odometry with some frame_id and child_frame_id to an output odometry with a new frame_id and child_frame_id
- Can take an input odometry and publish an output odometry with a frame_id and child_frame_id that are just renamed, not actually transformed
- Can publish the output odometry with a ros::Time::now() time stamp.
- Can publish a tf for an odometry message
- Can publish a stabilized tf (zero pitch and roll) for an odometry message

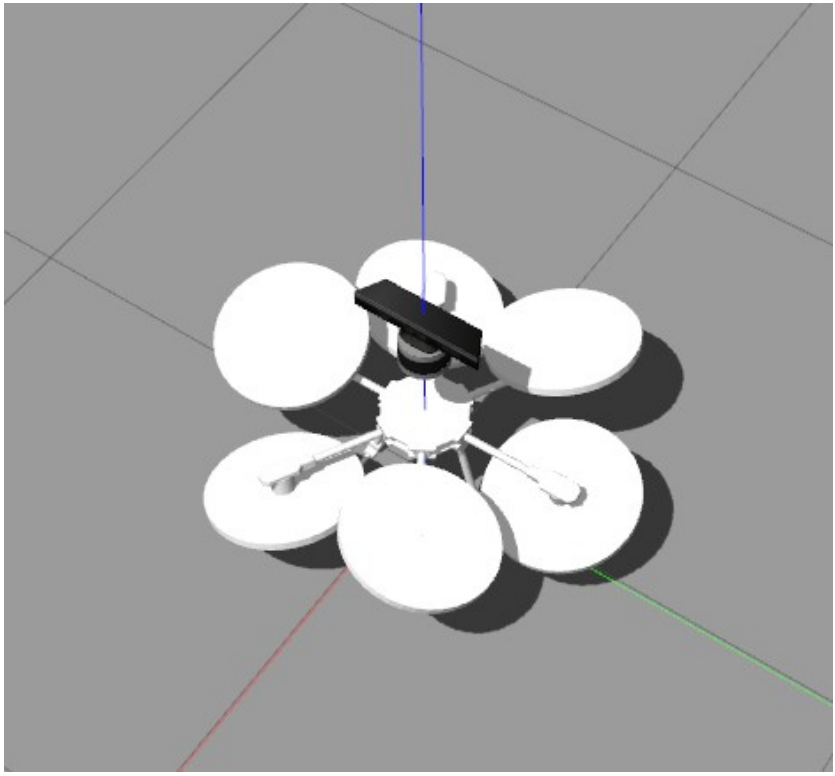
Core Odometry Transform

- Example: The gazebo plugin libgazebo_ros_p3d.so publishes a ground truth odometry message to the /uav1/ground_truth/state. The velocities are actually in the “world” frame, but the plugin mislabels them to be in the “base_link” frame
- Core odometry transform is used to rename the frame_id and child_frame_id



Core Gazebo Sim

- Contains a gazebo plugin which controls the velocity of a model. Works with gazebo's physics engine so collision still works.
- Contains urdf macros for a few common sensors:
  - RGBD camera
  - Stereo camera
  - Planar lidar
- Separate repo contains a simulated velodyne
  - Custom version of the velodyne_simulator package that fixes a bug where texture URL is incorrect
  - Adds configurable mass



Core Gazebo Sim

- Makes it easier to add sensors in a few lines of urdf

```
<!-- Add an rgbd camera -->
<xacro:include filename="$(find core_gazebo_sim)/urdf/rgbd_camera.urdf.xacro"/>
<rgbd_camera name="rgbd_camera" parent="base_link">
  <origin xyz="0 0 0.05" rpy="0 0 0" />
</rgbd_camera>

<!-- Add a stereo camera -->
<xacro:include filename="$(find core_gazebo_sim)/urdf/stereo_camera.urdf.xacro"/>
<stereo_camera frame_name="front_stereo" parent="base_link">
  <origin xyz="0 0 0.05" rpy="0 0 0" />
</stereo_camera>
```

Base Main Class

- Functions to override
  - initialize(): This is where you should set up ROS related objects like publishers and subscribers and check for ROS parameters.
  - execute(): This is where your main program logic should go. It gets called in a loop at a rate set by the execute_target parameter in your node's namespace.
  - ~Destructor(): Do cleanup, like memory deallocation etc., in the destructor.
- get_node_handle/get_private_node_handle
  - Easier to transition between nodes and nodelets
    - Nodes: ros::NodeHandle / ros::NodeHandle("~")
    - Nodelets: getNodeHandle() / getPrivateNodeHandle()

Base Main Class

- HealthMonitor
  - monitor.tic(“name”);
  - Code you want to time…
  - monitor.toc(“name”);

- [print_time_statistics]: Name Avg Hz Target Hz Avg ms Min ms Max ms Calls
- [print_time_statistics]: execute 5e+06 1 0.0002 0 0.003 15
- [print_time_statistics]: name 4e+06 1 0.03 0.001 0.5 80

- <param name=”name_target” value=”20” />
  - Will print error if code between tic and toc is taking longer than 1/20

Namespacing

In c++ code:

```
ros::Subscriber(“topic1”, 1, callback);
ros::Subscriber(“/topic2” 1, callback);
```

In launch file:

```
<launch>
	<group ns=”uav1”>
		<node name=”pid_control” pkg=”drone_controller” type=”pid_controller”>
			<param name=”P” value=”1.0” />
		</node>
	</group>

	<node name=”pid_control” pkg=”drone_controller” type=”pid_controller”>
		<param name=”P” value=”1.0” />
	</node>
</launch>
```

Namespacing

rosnode list:
/uav1/pid_control
/pid_control

rostopic list:
/uav1/topic1
/topic1
/topic2

rosparam list:
/uav1/pid_control/P
/pid_control/P

In c++:
ros::NodeHandle nh;
ros::NodeHandle pnh(“~”);

nh.param(“P”, 0.0); // incorrect
nh.param(“/pid_control/P”, 0.0); // incorrect, but works outside namespace if node name is pid_controller

pnh.param(“P”, 0.0); // correct

Namespaceing

- Global namespace topic examples
  - /rosout
    - ROS_INFO, ROS_ERROR_STREAM, etc...
  - /tf and /tf_static
    - tf::TransformListener and tf::TransformBroadcaster

How to use the stack

- Getting the code and building
- Running the examples
- Launch file structure

Getting and Building the Code

- core_central is the main package https://bitbucket.org/castacks/core_central/src/master/
- Create a workspace, clone core_central, choose your platform's rosinstall file, and build
- Extra step for building the PX4 firmware

Gazebo Sim

```
mkdir -p ws/src
cd ws/src
git clone git@bitbucket.org:castacks/core_central.git
ln -s core_central/rosinstalls/sim.rosinstall .rosinstall
wstool up
cd ..
catkin build
```

PX4

```
mkdir -p ws/src
cd ws/src
git clone git@bitbucket.org:castacks/core_central.git
ln -s core_central/rosinstalls/px4.rosinstall .rosinstall
wstool up
cd Firmware
make px4_sitl_default gazebo
cd ../..
catkin build
```

DJI

```
mkdir -p ws/src
cd ws/src
git clone git@bitbucket.org:castacks/core_central.git
ln -s core_central/rosinstalls/dji.rosinstall .rosinstall
wstool up
cd ..
catkin build
```

Getting and Building the Code

- Create branches for project specific modifications
- The master branch is locked
  - Send me a pull request

Running the Examples

- https://youtu.be/w_rxTrj0Io4 shows the example running
- There are examples of a drone in an empty world, drone in a town environment using lidar, drone in a town environment using disparity, and two drones in a town environment, one using lidar, one using disparity.
  - mon launch core_central <platform>_example_empty.launch
  - mon launch core_central <platform>_example_lidar.launch
  - mon launch core_central <platform>_example_disparity.launch
  - mon launch core_central <platform>_example_multiple.launch
- The Gazebo and PX4 version both can run in gazebo. DJI must be connected to a real drone to run the sim which doesn’t simulate lidar/cameras, so the examples do not apply. mon launch core_central dji_sim_drone.launch will work to control the simulated and real drones.

Launch File Structure

- In core_central/launch

- common/
  - autonomy.launch
    - Launches everything discussed in the architecture section except the pose/velocity controllers and state estimate as these are usually platform specific
  - visualization.launch
    - Launches the base station GUI and rviz

- <platform>/sim/
  - <platform>_sim_drone.launch (Main launch file for a drone)
  - <platform>_sim_state_estimation.launch
  - <platform>_sim_control.launch
  - <platform>_spawn_drone.launch*
  - <platform>_example_empty.launch*
  - <platform>_example_disparity.launch*
  - <platform>_example_lidar.launch*
  - <platform>_example_multiple.launch*
  - *Doesn’t include DJI, since it can’t be simulated in gazebo

Launch File Structure

- <platform>_sim_drone.launch structure

```
<launch>

  <arg name="robot_name" default="uav1" />

  <group ns="$(arg robot_name)">
    <!-- spawn a drone -->
    <include file="$(find core_central)/launch/gazebo/sim/gazebo_spawn_drone.launch" pass_all_args="true"/>

    <!-- run state estimation -->
    <include file="$(find core_central)/launch/gazebo/sim/gazebo_sim_state_estimation.launch" pass_all_args="true"/>

    <!-- run control -->
    <include file="$(find core_central)/launch/gazebo/sim/gazebo_sim_control.launch" pass_all_args="true" />

    <!-- autonomy -->
    <include file="$(find core_central)/launch/common/autonomy.launch" pass_all_args="true" />

    <!-- visualization -->
    <include file="$(find core_central)/launch/common/visualization.launch" pass_all_args="true"/>
  </group>

</launch>
```

Launch File Structure

- Example launch file
  - First start gazebo with a world file
  - Next add drones to the world with the main arguments for defining it
    - **robot_name** – All topic names, node names, and tfs will be prefixed with this name so multiple drones can be created without conflict
    - **drone_interface** – either GazeboInterface, PX4NoTiltInterface or DJIInterface
    - **map_representation** – either DisparityMapRepresentation or PointCloudMapRepresentation
    - **x, y, z** – the position to spawn the drone

```
<launch>

  <!-- gazebo simulation -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find core_gazebo_sim)/worlds/town.world"/>
  </include>

  <include file="$(find core_central)/launch/gazebo/sim/gazebo_sim_drone.launch">
    <arg name="robot_name" value="uav1" />
    <arg name="drone_interface" value="GazeboInterface" />
    <arg name="map_representation" default="PointCloudMapRepresentation" />

    <arg name="x" value="0" />
    <arg name="y" value="0" />
    <arg name="z" value="0.2" />
  </include>

  <include file="$(find core_central)/launch/gazebo/sim/gazebo_sim_drone.launch">
    <arg name="robot_name" value="uav2" />
    <arg name="drone_interface" value="GazeboInterface" />
    <arg name="map_representation" default="DisparityMapRepresentation" />

    <arg name="x" value="0" />
    <arg name="y" value="3" />
    <arg name="z" value="0.2" />
  </include>

</launch>
```

Exercise

- Implement the Autonomously Explore functionality
  - Get familiar with the sim and set up the rqt gui
  - Created a node that publishes a Bool indicating whether a waypoint has been hit
  - Modify the behavior executive to implement the “Hit Waypoint” condition using the Bool
  - Modify the behavior tree to make the drone takeoff, explore until the “Hit Waypoint” condition is true, then land

Exercise

- Declarations

```
// variables
bool got_waypoint, got_odom;
tf::Vector3 waypoint, odom;
bool hit_waypoint;

// publishers
ros::Publisher hit_waypoint_pub;

// subscribers
ros::Subscriber waypoint_sub;
ros::Subscriber odom_sub;
tf::TransformListener* listener;

// callbacks
void waypoint_callback(geometry_msgs::PoseStamped msg);
void odom_callback(nav_msgs::Odometry msg);
```

Exercise

- Initializing

```
ros::NodeHandle* nh = get_node_handle();
ros::NodeHandle* pnh = get_private_node_handle();

// init variables
got_waypoint = false;
got_odom = false;
hit_waypoint = false;

// init subscribers
waypoint_sub = nh->subscribe("custom_waypoint", 1, &Exploration::waypoint_callback, this);
odom_sub = nh->subscribe("odometry", 1, &Exploration::odom_callback, this);

// init publishers
hit_waypoint_pub = nh->advertise<std_msgs::Bool>("hit_waypoint", 1);
```

Exercise

- Checking distance between the drone and waypoint

```
if(got_odom && got_waypoint){
  ROS_INFO_STREAM("distance: " << waypoint.distance(odom));
  if(!hit_waypoint)
    hit_waypoint = waypoint.distance(odom) < 4.0f;

  std_msgs::Bool hit_waypoint_msg;
  hit_waypoint_msg.data = hit_waypoint;
  hit_waypoint_pub.publish(hit_waypoint_msg);
}
```

Exercise

- Callbacks

```
void Exploration::waypoint_callback(geometry_msgs::PoseStamped msg){
  got_waypoint = true;
  waypoint = tf::Vector3(msg.pose.position.x, msg.pose.position.y, msg.pose.position.z);
}

void Exploration::odom_callback(nav_msgs::Odometry msg){
  got_odom = true;
  odom = tf::Vector3(msg.pose.pose.position.x, msg.pose.pose.position.y,
msg.pose.pose.position.z);
}
```

Exercise

- CMakeLists.txt and package.xml

Add nav_msgs and geometry msgs to find_package() in CMakeLists.txt

Add to package.xml:
<build_depend>geometry_msgs</build_depend>
<build_depend>nav_msgs</build_depend>

<build_export_depend>geometry_msgs</build_export_depend>
<build_export_depend>nav_msgs</build_export_depend>

<exec_depend>geometry_msgs</exec_depend>
<exec_depend>nav_msgs</exec_depend>

Exercise

- https://bitbucket.org/castacks/exploration_tutorial_week_2020/src/master/
- https://bitbucket.org/castacks/core_behavior_executive/src/tutorial_week_2020/
- https://bitbucket.org/castacks/core_gazebo_sim/src/tutorial_week_2020/
- https://bitbucket.org/castacks/core_central/src/tutorial_week_2020/

Thanks!