ROS Setup

1. Camera Calibration
   To identify image correspondences in the camera frame, an intrinsic calibration process is needed. This facilitates the process of identifying features in 3D space that are projected into the image plane and vice versa.

   As a first step, a calibration target will be used to identify 2D/3D correspondances. We will use a checkerboard pattern as the one shown below with predetermined dimensions.

   To identify the intrinsic parameters for the camera, the [ROS calibration package]() based on OpenCV will be used. The process of installing it and running the node is described below.

   Install the camera_calibration package

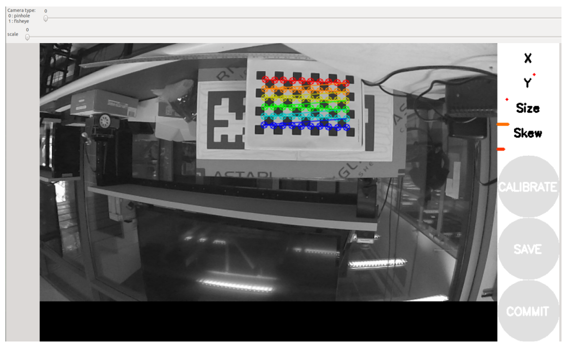
   ```
   sudo apt-get install ros-melodic-camera-calibration
   ```

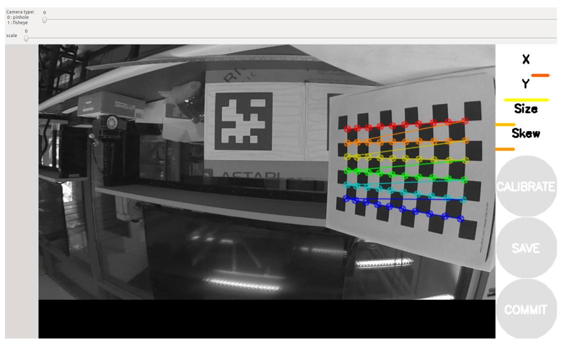
   Run the node using a 7x11 checkerboard (with square size=22mm)

   ```
   rosrun camera_calibration cameracalibrator.py --no-service-check --size 6x10
   --square 0.022 image:=/jetbot_camera/raw camera:=/jetbot_camera
   ```

   A window should open and display image information. If the checkerboard is within the field of view, keypoint features should be superimposed on the frame. Note: if note detections are generated, ensure that there are consistent lighting conditions and that the dimensions of the board are correct.

   To gather enough 2D/3D correspondences for calibration, move the board along various parts of the image frame slowly and obtain various perspectives over different distances from the camera. Press the CALIBRATE button when the button turns green (try gathering over 40+ correspondances). It may take a few seconds for the parameters to be estimated; when the parameters are printed in the terminal, the calibration files (tarball) can be saved by clicking on the SAVE button. By default, the calibration files are saved in /tmp.

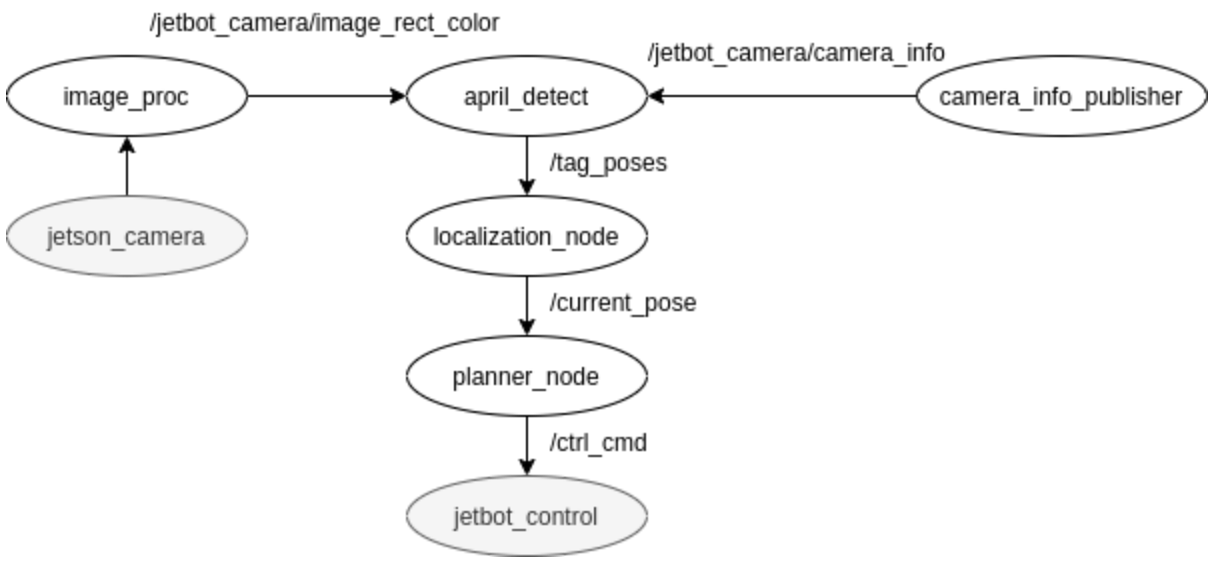




2. AprilTags for Pose Estimation
   To localize our robot in map coordinates and correct errors associated with dead reckoning, AprilTags will be used for pose estimation. However, before we can detect the markers present in image space and estimate the rigid body transformation between the camera and the marker, we will need to undistort our image using the distortion coefficients (d) estimate through the calibration process.

   a. Update jetbot_ros package
      i. Backup any files within your jetbot_ros package
      ii. Pull latest commit from Github [repository]
      iii. Rebuild package
```
cd ~/workspace/catkin_ws && catkin_make --pkg jetbot_ros
```
   b. Extract and Copy calibration parameters to jetbot_ros package
      i. Extract calibration parameters from YAML files
```
$ tar xvzf calibrationdata.tar.gz
$ mv ost.yaml
~/workspace/catkin_ws/src/jetbot_ros/scripts/camera_intrinsics.yam
l
```
   c. Install AprilTags library
      i. `pip2 install --user apriltag`
   d. Run april_localization node and dependency nodes
      i. Build navigation_dev ROS package
```
cd ~/workspace/catkin_ws/src
git clone https://github.com/AftermathK/navigation_dev.git
cd ~/workspace/catkin_ws/
catkin_make --pkg navigation_dev
source devel/setup.bash
```
      ii. Run camera parameter publisher
```
rosrun jetbot_ros camera_info_publisher.py
```
      iii. Rectify distorted image data using image_proc node
```
ROS_NAMESPACE=jetbot_camera rosrun image_proc image_proc
```
      iv. Detect and determine transformation between camera and marker *i*
```
rosrun navigation_dev april_detect.py
```
      v. To visualize AprilTag detections and verify that the april_detect node is operational, you can use RViz to visualize Image data being published under `/april_detections`. Additionally, the rigid body transformations can be printed using `rostopic echo /tag_poses`

3. Sample code
   To facilitate integration of your code with subsequent assignments, ROS node skeletons have been provided for the localization, planner, and control modules. These are provided for reference to help you get started; however, you are welcome to experiment with a different design strategy. Our design is outlined in the figure below. The skeleton code can be executed as follows `roslaunch navigation_dev navigation.launch`


/jetbot_camera/image_rect_color
image_proc
april_detect
/jetbot_camera/camera_info
camera_info_publisher
jetson_camera
/tag_poses
localization_node
/current_pose
planner_node
/ctrl_cmd
jetbot_control