Xihelm Test Report

I wanted to thank you for the opportunity to complete the test you sent me. I thoroughly enjoyed solving it, and it was a fun and great experience for me. .

As per the test's requirements, I have developed a solution to STOP ALL MOTORS when "Bad Objects" are detected in any of the multitudes of cameras. Here's a brief report and discussion of my solution:

Solution:

The code is designed considering scalability for multiple cameras & flexibility.

- **Camera Publisher**:
    1. I have developed a Camera Stream Publisher that reads the RTSP link and respective camera ID from the MongoDB database. For scalability for multiple cameras, you just need to add an RTSP link to this database.
    2. It then publishes the stream based on the unique topic generated from the camera ID.
    3. You can view these streams using “rqt_image_view”.

- **Object Detection:**
    1. This module subscribes to all the above-published topics and performs object detection on them.
    2. In case any object is detected in any of the streams, the module publishes the Motor status on a separate ROS Topic.

- **Motor Control:**
    1. This turns the motor on or off and sets the Twist message accordingly based on the Motor Status published by the Object Detection module

Assumptions :

- In real-time, there would be an RTSP link, but for now, I’ve considered the path of multiple video files and have processed these videos

- Since there was no video for detection, I used a video in which a person walks into the frame from left to right and afterward I added black frames for a few seconds. This is a similar analogy to the video from the Conveyer belt camera.

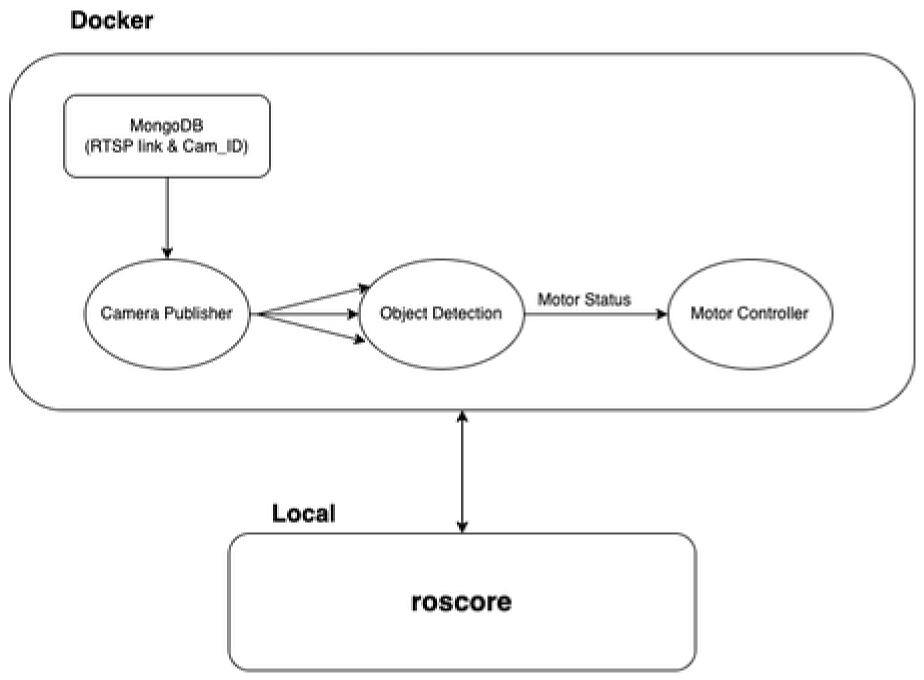
- *Object Detection* - I’ve not focused much on doing ML for object detection. I’ve just developed an algorithm:
    1. `Gray image = cv2(BGR2GRAY)`
    2. `Foreground background separation.`
    3. `Make it blur`
    4. `Find Contours.`
    5. `If len(Contours) >=1:`
        a. `You can assume that BAD object is detected.`
    6. ***It’s like when a person is there in a frame it should detect the object and TURN MOTORS OFF!***

- Motor supports `Twist` msg format.

Architecture:

For scalability purposes, I’ve designed three different nodes. Having 3 separate nodes would help to debug respective modules and scale them accordingly. Of course, having a single node for object detection and camera publisher would decrease time complexity but would become hard to handle at scale.

Hence, I’ve selected the **linear** approach for this task.



- **MongoDB**: The camera publisher will read rtsp link from the Mongo database and would publish a stream accordingly. I've written a utility (db_utils.py) to manage the database. (Insert or delete data).

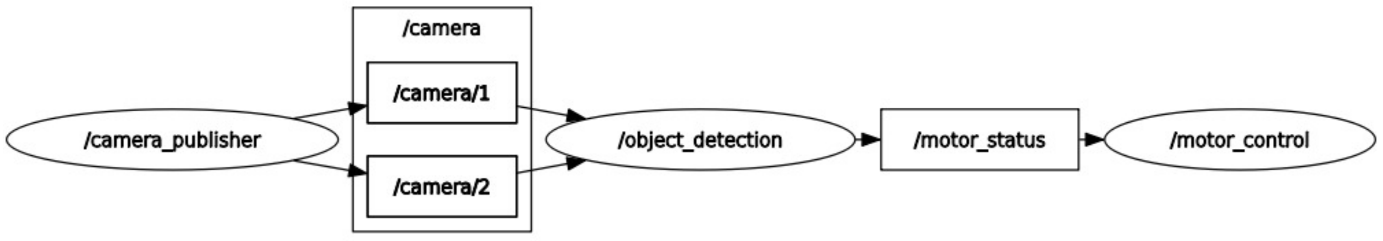
- **Camera Publisher:**
  - Each camera has each image topic published.

- **Object Detection :**
  - Checks the list of topics published every `5 sec.` If a new camera is added, it would subscribe to it and do object detection.
  - If required objects are detected, it publishes Boolen data to Ros topic.
  - It would loop over a list of topics (updated every 5 sec) and pass on for object detection.

- **Motor Controller:**
  - Just to make debugging easier, I've made up a separate Motor Controller node. So in the future, if any module fails, it's easy to analyze.

- **Docker**: Entire system is been deployed using Docker as a Node, which would use Roscore network running on a local machine. This would be easier for maintaining other features for ROBOT and not messing all of them together.

- **RQT Graph:**



Limitations & Further Improvements:

(Where My Code can break)

- **Object Detection Node:**
  - This node is reading each publisher in series and passes them for object detection.
  - If there are 10,000 Cameras, it would take some time, and during this phase and if any bad object passes it would break
  - _Solution_:
    - Multi-processing should solve this problem
    - We can use **thread pooling** from python and start the threads in parallel for each camera topic and do processing the frames for detection.

- **Frequency of Object Detection:**
  - Generally, DL Models will detect objects at every instant. This would cause frequent turning ON & OFF of motors.
  - _Solution_:
    - We can control the rate of nodes using RATE function in ROS. Adding this in the Motor controller/ object detection node could control the rate at which it should listen and do the switching.

- **Docker Memory:**
  - Since we are using MongoDB + ROS inside the same docker, memory would get very high.
  - _Solution_:
    - **Ideally, MongoDB should run in different dockers. We can get the IP of this MongoDB server and interact with this from our ROS Docker.**
  - *The reason I selected to add mongoDB in the same docker was, the MongoDB docker requires some other PC Configuration, unfortunately, my PC doesn't support that. Also, MongoDB guys have not solved this issue yet!*