Jon Rippe
PDAQS
Progress Report 21.9.1

Issues Addressed

- Video Processing
- Object Tracking
- Overall Architecture

Object Tracking Across Frames

The object tracking modules worked well for displaying tracked objects in the camera's frame, however they were not structured in such a way as to allow easy data logging or tracking between cameras. Object tracking will be done using lidar points rather than camera frame information. Object detection works and will remain mostly untouched.

Notable Progress

A new PDAQ class has been created that's responsible for coordinating data received from the LiDAR stream and the camera streams. The PDAQ class will also be responsible for object tracking. The PointFilter class has been removed and its functions integrated into the PDAQ class. High-level functionality has been added and low-level functionality has been stubbed out. Several new object types and modules have been integrated into the design. Figure 1 on the next page shows the new workflow.

The VelodyneManager run() function has been threaded and will now output frames to a frame buffer for the PDAQ class to get.

New Modules

```
VelodyneManager(VLP type, pcap file, output folder, parameters)
       .run() -> Action: extracts pcap data and saves frames to output folder AND/OR queue
       .start() -> Calls run() as a thread

Camera(video path, output path, params, azimuth, FOV, xyz offset)
       .__init__() -> starts video stream
       .get_frame() -> gets frame from buffer
       .write_frame(frame) -> writes frame to output
       .shutdown() -> kills video stream

PDAQ(pcap path, output path, List[cameras], params)
       .run() -> do all the things then kill all the things and save when done
       .draw_overlay(frame, points, objects) -> draws points and object data on a frame
       .get_object_data(object, points, lidar_points) -> determines object center and size in 3D
       space based on object center and size in camera frame.
       .transform_points(points, camera) -> Transforms points to match camera pixels

TrafficDetection(width, height)
       . traffic_detections(frame) -> detects objects in frame.  Returns class, center, radius.

Mnet -> Neural network class for object classification

TrackableObject(ID, center, radius, timestamp, label, attribute smoothing, frame persistence)
       .predict_position(timestamp) -> predicts object position based on current data
       .is_collided(center, radius, label, error margin) -> determines if passed in parameters are
       the "same" object
       .update_object(center, radius, timestamp) -> updates object based on new data
       .calculate_attributes(timestamp) -> calculates current velocity and acceleration
       .is_out_of_frame() -> has been out of frame for more than desired check time
```

Next Steps

Stubbed functions need to be filled in. They are mostly low-level functions and will be time consuming.

Timing needs to be synchronized. Perhaps write code to find the latest start time of all the cameras and LiDAR, then purge all frame buffers up to that time. Also need to consider accounting for LiDAR FPS and camera FPS being different. Probably use LiDAR timestamp as a master timestamp and decide which camera frame to use based on master.

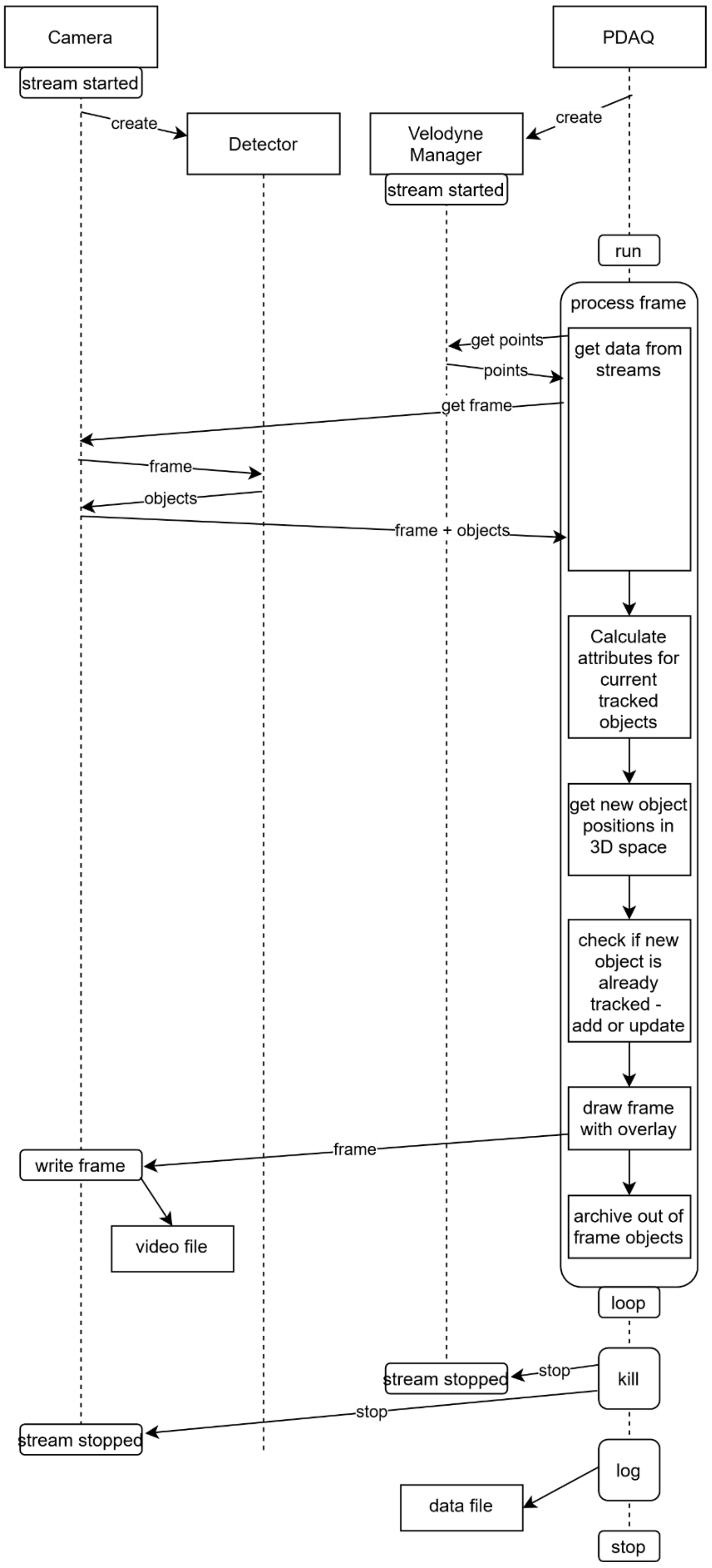

Figure 1: Basic Workflow