Avoidance, Detection, and Tracking in Low-Power Autonomous Systems

William Luer

Silvia Zhang, Project Advisor
Ron Cytron, Committee Member
Ayan Chakrabarti, Committee Member

May 2nd, 2018

Computer vision algorithms are typically reserved for platforms that can handle the computational workload needed to process the huge amount of data in images and videos. The recent surge in artificial intelligence, machine learning, and computer vision have guided the development of powerful processors that can quickly and more efficiently handle the computationally intensive algorithms. For this project, I aimed to go against the grain and implement computer vision and artificial intelligence on a Raspberry Pi, a low-power IoT device that is the on-board processor for a small autonomous vehicle project called the PiCar.

The first part of the project was the development and implementation of a real-time control algorithm using optical flow and machine learning to successfully navigate randomly generated obstacle fields. The processing was done entirely on a Raspberry Pi 3 and the video stream was provided from the standard Raspberry Pi camera module. The algorithm worked in the following manner:

1. Read images from video stream
2. Detect features using Shi-Tomasi corner detection
2. Calculate optical flow vectors
3. Calculate time to contact (TTC) for each tracked point (x,y)
4. Cluster three dimensional data (x,y,TTC) using DBSCAN
5. Sort clusters by lowest TTC
6. Calculate servomotor angle and motor PWM
7. Send signal to motor-controller via SPI

The algorithm ran at 8-15 frames per second and varied with the number of tracked points. As the number of tracked points increased, so did the processing time needed at each step throughout the algorithm. The algorithm and the PiCar platform were presented and selected as a finalist at the Xi'an Jiatong University Silk Road Robotics Innovation Competition during the summer of 2017.

Although successful, there are many limitations to the aforementioned algorithm. The algorithm was developed to be used in an environment with static obstacles and no end destination. It's sole goal was to drive and not hit anything, i.e. obstacle avoidance. To combat these limitations, I began developing object classification and tracking algorithms with the goal of detecting objects and acting appropriately.

Classification is a cumbersome process and the amount of time needed to process a single image is unrealistic for real-time applications on low-power devices. Therefore, single shot detection (SSD) was used on the first frame from a video stream, which took about 1.5 seconds to run on the Raspberry Pi, and the resulting bounding box was passed to a tracking algorithm that ran at approximately 30 FPS but would often drop to 10 FPS. Initially, the provided OpenCV tracking modules were used (KCF and MedianFlow). These algorithms were a black-box routine and although they were functional, it was difficult to understand the inconsistencies in the processing time of the algorithms.

To investigate the variance in the tracking algorithms and provide a basis for Professor Silvia Zhang's research on power management in mobile robotics, I dove further in to the pure tracking component and developed a testing platform for these tracking algorithms. With this testing platform, users can specify the keypoint detection method, the feature matching method, the video source, and whether or not they would like to generate a performance report. With customization, users can experiment and determine which tracking method is best for a certain application/situation.

Overall, five different keypoint detection methods were implemented: random sampling, SIFT, SURF, ORB, FAST, and Shi-Tomasi. Three feature matching methods were implemented: optical flow, FLANN, and brute-force. To generate a performance report and calculate error, there must exist true data for which the tracking algorithms can be compared against. There is a script that can be run in which users manually select the middle of the object they desire to be tracked, frame by frame. This data is stored in a CSV file and used in the root mean square error and mean absolute error calculations. In addition to the calculated error, the performance report also details the average FPS, whether or not a tracking failure occurred, the average number of tracked points and four distinct plots: error vs number of tracked points, FPS vs number of tracked points, FPS vs frame number, and the time taken by the keypoint and feature matching methods vs frame number.

To use this platform the user would follow the steps below:

1. Record video (recordVideo.py)
2. Manually generate true tracking data (clickTrueData.py)
3. Run tracking customization via command line (TrackingManager.py)
4. Manually analyze performance report

TrackingManager.py command line usage can be seen below:

```
python3 TrackingManager.py [-h] [-e] -s SOURCE -k KEYPOINT -m MATCHER
```

Github link:
https://github.com/xz-group/PiCar

Xi'an Jiatong University Silk Road Robotics Innovation Competition Submission Video:
https://www.youtube.com/watch?v=2QJigzSVVe4

HEC-TV Video Link:
https://www.youtube.com/watch?v=4JIqfRzICwc