UNIVERSITY OF PENNSYLVANIA

ESE SENIOR DESIGN TEAM 24: DARKMODE

ISD PROJECT

Spring 2022 Report

Computer Vision Algorithms for Event-Based Cameras Applied to Autonomous Low-Light Driving

Authors:
Vinay Senthil, CIS & WH, vinayksk@seas
Anish Neervannan, CIS & WH, anishrn@seas
Neil Chitalia, CMPE & WH, nchitali@seas
Keshav Vedula, MEAM & WH, kvedula@seas
Philip Sieg, MEAM, philsieg@seas

Advisor:
Prof. Pratik Chaudhari
pratikac@seas

April 30, 2022

Contents

II Executive Summary

A major roadblock to fully autonomous driving becoming a mainstream technology is its reliability in night time and low light scenarios. Current leaders of the autonomous movement such as Tesla have gravitated towards camera and computer vision based models to perform the bulk of their autonomous control. However, these systems are currently limited by the hardware's ability to give detailed image data to these models in low environments with low light and motion blur. To that end, we have developed a suite of perception and mapping models using event based cameras (in tandem with conventional cameras), which have significantly better performance in fast-paced environments because of the cameras' increased light sensitivity and variable pixel refresh rate. To demonstrate the capabilities, we have built a prototype inspired by a F1Tenth car.

Our solution is segmented into three main components: perception and mapping, planning and controls, and physical and software integration onto the F1Tenth Car. The primary task of our perception and mapping team is to process event based data and perform Simultaneous Localization and Mapping (SLAM) in order to create maps. Additionally, we implemented object detection and optical flow as an introduction to using learning models with event based data. Our planning and controls teams uses this map of the environment as an input to plan a path in this environment, and command the motors such that this path is closely followed. Finally, we integrated these components into the construction of the autonomous F1Tenth car to account for challenges such as noise in the maps and communication (computational) constraints.

Currently, we can generate a map of our surroundings and localize the vehicle's position using data from a RGB camera and ORB-SLAM3. Leveraging the data from the event based camera, we perform optical flow and object detection. From the sparse occupancy grid generated by SLAM, we collapse the information into a 2D map of obstacles and free space. Utilizing this map, we perform path planning to reach a goal point from the vehicles current position. Our controller then sends motor commands to the vehicle so that it follows the generated path. We upgraded a Traxxas RC car with new hardware and a computational suite that has served as our testing platform.

III Overview and Motivation

For every million miles travelled, AV's experience an average of 9.1 crashes. This is more than double the figure for traditional vehicles, which experience an average of 4.1 crashes. The National Institute of Health estimates that a third of all AV crashes are due to sensor or perception failure. A major roadblock to fully autonomous driving becoming a mainstream technology is its reliability in night time and low light scenarios. Current leaders of the autonomous movement such as Tesla have gravitated towards camera and computer vision based models to perform the bulk of their autonomous control. However, these systems are currently limited by the hardware's ability to give detailed image data to these models in low environments with low light and motion blur. To that end, we aim to develop a suite of perception and mapping models using event based cameras (in tandem with conventional cameras), which have significantly better performance in fast-paced environments because of the cameras' increased light sensitivity and variable pixel refresh rate. Our goal is to adapt current mapping and planning algorithms for event-based cameras on an F1Tenth RC car with an asynchronous paradigm.

IV Technical Description

Our overall solution can be partitioned into three main components: perception and mapping, and planning and controls, and physical and software integration. The primary goal of the first component is to generate an accurate map of the vehicles surroundings, while localizing its position within that map. The second component uses this map to create a path from its current position towards a goal. To follow this path, we calculate control outputs that will be sent to the motors of the vehicle. The final component integrates the algorithms from the first two with the physical car. Our team accomplishes this through utilization of ROS, a middle-ware used for robotic systems. The third component also contains all hardware modifications made to the vehicle.

IV.I Perception and Mapping

The primary goal of our perception and mapping team is to be able to process event based data and perform Simultaneous Localization and Mapping (SLAM) in order to create maps that can be used for path planning and controls. We aim to do this by implementing ORB-SLAM3 which uses RGB input to generating a 3-D sparse occupancy grid of the surroundings, and then convert this into a 2-D map to be used by the path planning and controls team.

As secondary and tertiary goals, we also aim to implement traditional computer vision algorithms such as object detection and optical flow using event-based DVS data for real-time dynamic collision avoidance. This extension is more applicable to our ethical considerations; for example, distinguishing between pedestrians and other obstacles on the road and making real-time decisions to avoid accidents.

IV.I.I SLAM

Depending on the hardware, SLAM algorithms can complete 2-10 iterations per second, updating the map at a relatively low frequency. Our goal in integrating an event based paradigm to the mapping goal was to provide cars with dynamic collision avoidance. By receiving event-based data in between the frames of RGB images, we are better equipped at finding objects that are in a collision course with the car or suddenly enter the frame. The major constraints that we faced in terms of building an autonomous vehicle were the cost of the sensors, weight of the components (compute units, DVS, power), real-time computing deadlines, and energy usage. Modifying traditional autonomous systems to use a DVS saves cost in comparison to LIDAR units, saves computation (both monetary and in compute time) because only updates have to be processed asynchronously, reduces weight, and increases compactness of the sensor suite making the DVS and accompanying algorithms ideal for AVs operating in high speed or low light scenarios. Initially, we wanted to a pure event-based SLAM, but this requires an extensive redesign of traditional algorithms from first principles. Thus, our main objective is to modify traditional perception and mapping algorithms to incorporate dynamic collision avoidance, while saving on time and energy.

Our first step was to generate a map using ORB-SLAM and a traditional RGB camera. Since ORB-SLAM is a well documented algorithm, we can quickly generate a map that can be used by the Path Planning and Tracking team, while we set up the DVS sensor. ORB are binary features invariant to rotation and scale and can be computed in real time without the use of a GPU. ORB-SLAM computes these features every 33ms and computes the pose from the previous frame. If the changes are significant, the current frame is classified to be a keyframe. By tracking the pose of the car as well as the movement of ORB features through keyframes, we can generate the position of these frames in the global state space. Over time, a survival of the fittest keyframe selection algorithm allows us to remove redundant

keyframes and close loops. We will use an existing implementation of ORB-SLAM and a webcam and drive the car around our house to generate our map.

The result of ORB-SLAM is a 3D sparse occupancy grid, where points in space are determined to be occupied after passing a specified probability of occupancy. However, the path planner can't operate on this 3D matrix and is currently designed to work on a 2D binary map. Furthermore, the sparse points must be converted into defined borders so that the planner does not generate a path through walls. From the sparse points, we must determine the borders of walls. We leveraged an existing algorithm called Bresenham's line algorithm that allows us to convert the 3D matrix to a 2D map in real time.

To implement dynamic collision avoidance, we will use the DVS data and the optical flow algorithm to determine if objects in the surroundings have relative velocities that intersect the car's current location and path. If so, the car either swerves, slows, or stops depending on the object's velocity.

IV.I.II Event Based Computer Vision

Although real-time collision avoidance was not a main focus of our project, we first implement object detection and optical flow as an introduction to using learning models with event based data. Both these features do not require the extremely high frequency of events given by the DVS and thus we were able to bin events into discrete time slices and still reduce computational requirements and get high confidence predictions. We hope to integrate these two features to perform dynamic object detection and velocity prediction in order to add to our path planning to do collision detection and avoidance. In regards to developing these algorithms, their structure mostly follows traditional RGB camera object detection and flow algorithms with a few changes in order to incorporate events.

To perform object detection on the events, we create a multi-layered neural network architecture that looks for high level features that persist across times to define them as events. The set of events, $\mathbf{E}$, is constructed from asynchronously streamed in events, i.e. $\mathbf{E} = \{e_i = (x_i, y_i, p_i, t_i)\}$ from which we want to learn a set of bounding boxes $\mathbf{B} = \left\{b_j^* = (x_j, y_j, l_j, h_j, t_j)\right\}$. To that end, we learn a detector, D that maps $\mathbf{E}$ to $\mathbf{B}$ where D only takes in events before time t to predict the bounding boxes at t. However, making a prediction for each event is too computational intensive and not required as the frequency of input stream is typically much higher than we need to detect objects. Instead we take a small interval Δt (which is less than the frame delay of a typical RGB camera) and accumulate all events in the interval before making a detection. While this improves the computational demand of the model, we found that there are many holes in the detector as it cannot detect apparently stationary objects because there are no new events coming in. Thus, we decided to incorporate some kind of memory to deal with cases of relatively stationary objects. To do so we save the internal state of the accumulated events using a learned weighting over time so that in areas with low density of new events, we are still able to make predictions.

At this point, our model is constructed as follows.

$$D(\mathbf{E}) = D\left(\{e_i\}_{t\in[t-\Delta t,t)}, h_{t-\Delta t}\right), \text{ where } h \text{ is computed recursively as follows} \quad (1)$$

$$h_t = F\left(\{e_i\}_{t\in[t-\Delta t,t)}, h_{t-\Delta t}\right) \text{ and } h_0 = 0 \quad (2)$$

Although our initial approach was to operate on sets of events, we found better results by creating a dense representation in the form of a tensor map. We then use a combination of convolutional neural networks containing ConvLSTM layers which incorporate memory and temporal relations to extract slow moving and time-persistent features in order to learn functions D and F. We followed the construction

of the object detection architecture proposed in *Learning to Detect Objects with a 1 Megapixel Event Camera* (Perot et al.). Finally, we feed our features into a bounding box regression head with a temporal consistency loss in order to learn and then predict the location and dimensions of the bounding boxes. The temporal consistency loss allows for another form of regularization where we focus on choosing objects that persist over time to avoid sporadic and noisy predictions of boxes that appear in only one time slice.

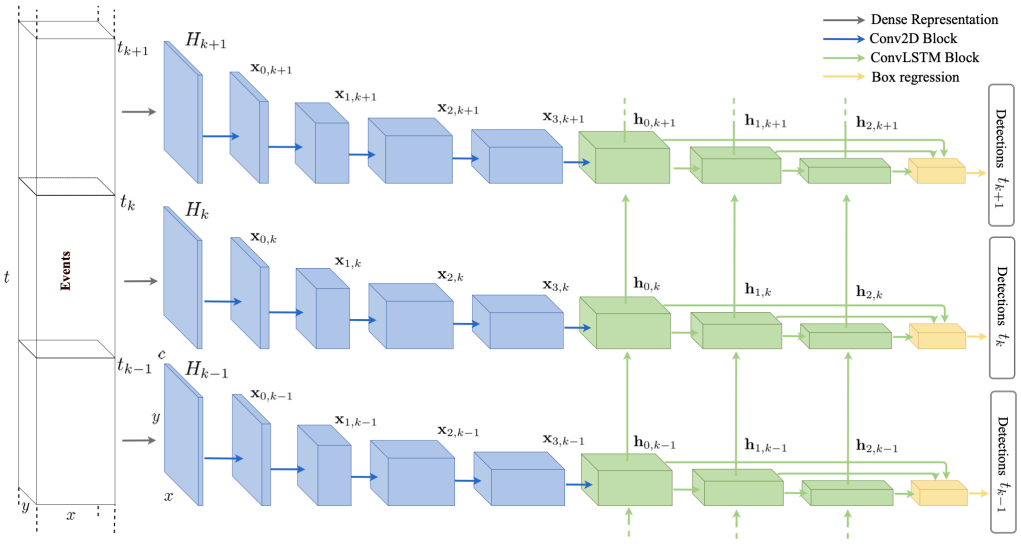


Figure 1: Proposed architecture from **Perot et al.** showing feed forward convolutional layers and ConvLSTM layers to extract low level features and high-level spatio-temporal patterns respectively.

Next, the optical flow algorithm was modelled after more traditional algorithms that work on a frame by frame basis. However, the main difference is that we use the event data stream to construct time surfaces for each time bucket. A time surface is a summarization of the recent history of events that maps the events from simple polarities to an exponentially decaying real number over time. We followed the implementation in *Optical Flow Estimation by Matching Time Surface with Event-Based Cameras* done by Nagata et al. using time surfaces to accumulate events over buckets and then estimate flow. To do so, we compute two loss functions for each pixel and predict the vector that minimizes that loss. The first component of the loss is the surface loss which is computed by taking the predicted velocity vector and using that to shift the pixel at position (x, y) to position $(x + v_x \Delta t, y + v_y \Delta t)$ and comparing that with the time surface shifted by Δt. We then add to that a smoothness regularization term which ensures the the new predicted velocity vectors do not vary sporadically and instead change slowly over time as we expect for object in the real world. The model is then tuned to determine the respective weights of each of the losses until we achieve high train/validation accuracy.

$$\min_{v} \left(L_{\text{smoothness}}(\mathbf{v}) + \lambda L_{\text{surface}}(\mathbf{v}) \right)$$

Lastly, we tuned the accumulation time Δt so that the optical flow computation could finish before the next iteration but also where it is small enough where the assumption of constant velocity in that time period is not violated. Running the algorithm with Prophesee's visualization software results in a strong visual confirmation that flow predictions are accurate.

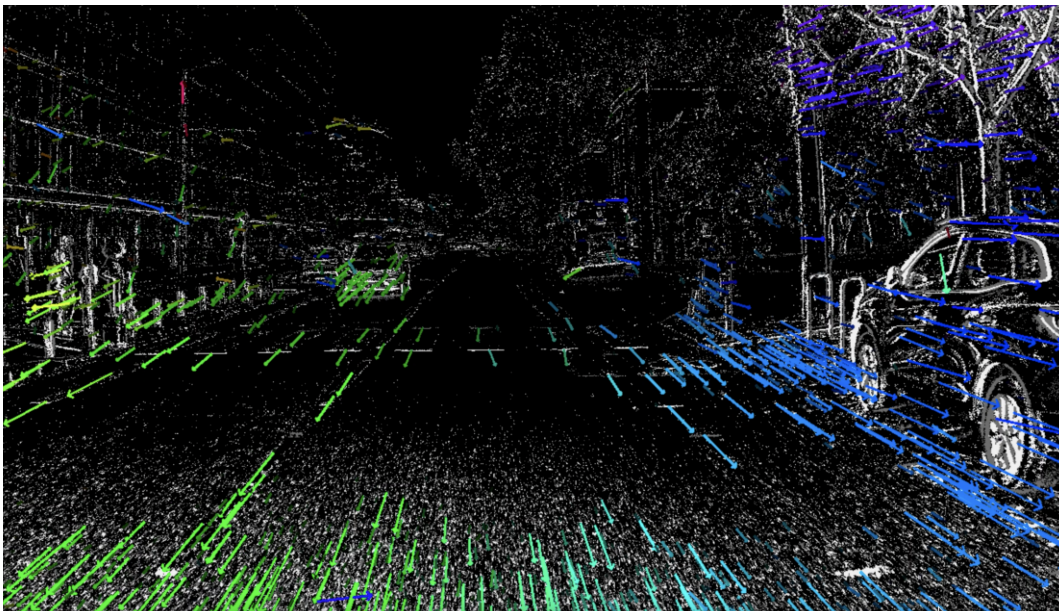


Figure 2: Output of optical flow algorithm on Prophesee's event based data visualing tool and Zurich DVS driving data

Other implementations that we considered and that exist in the current literature regarding event based optical flow include using adjacent pixel polarity and time surface values to estimate spatial gradients and then using traditional object detection on a reconstructed image but this would further underutilize the main benefit of sparse data and avoiding redundant computation.

IV.I.III Current State

After running into depth perception issues with SVO-SLAM, we opted to use ORB-SLAM. Using the Intel Realsense d435i camera, we can generate a map of the surroundings while the car under remote control operations. We are currently still debugging issues converting the 3D matrix into a 2D map in realtime. For the DVS, we have reconstructed multiple papers on object detection and optical flow and ran them on prerecorded datasets of driving and pedestrians. We have also created our own ROS package to process the DVS datastream and can run the optical flow and object detection models in real time using the feed from the DVS sensor.

IV.II Planning and Controls

The objective for the planning and controls team is to take as input a map of the environment from the perception and mapping team, plan a path in this environment, and command the motors such that this path is closely followed. Both the path planning and controller algorithms are largely subject to requirements derived from safety standards for autonomous vehicles. At a high level, the path planning algorithm should arrive at a collision free path in finite time, or the vehicle should immediately brake. This places an implicit constraint on the computational efficiency of the algorithm given that it will run onboard the vehicle. The generated path should approximately minimize some cost function – time, distance, or power consumption for instance – such the path is reasonably efficient. The controller

should command the motors to follow the path such that the deviation from the planned path remains below some preset threshold.

At this time, the National Highway Traffic Safety Administration (NHTSA) does not have specific performance standards to regulate Level 3 and Level 4 autonomous systems due to their novelty. However, NHTSA has published recommendations for states to regulate the testing process of such autonomous systems which are a relevant reference here. In particular, our path planning and controller algorithms should be designed to know and communicate when something has gone awry and there is a risk of collision. This can be accomplished by reporting when the path planning algorithm exceeds a set number of iterations for a single step, and when the controller algorithm measures a deviation from the planned path larger than some preset threshold. Under these exception cases, the vehicle should have a means of regaining control or immediately braking. Additionally, the path planning and controller algorithms should be designed such that a short time history of the planned paths and controller inputs are recorded and stored for post-mortem analysis in the event of a collision.

IV.II.I Path Planning

Given a map, the next stage of autonomous driving is to generate a collision free path from the current position to the goal. The first major consideration is whether to plan in the workspace or the configuration space. In order to avoid the need to translate obstacles to configuration space, we can do our planning in the "workspace" which is a two dimensional representation of the world. We will then use our controller to reach configurations which follow this path.

When it comes to path planning, there are many choices for algorithmic approaches. A popular option is A* because of its completeness. A* builds off of search algorithms which perform their planning in discretized space such as wildfire and breadth first. It expands Dijkstra's algorithm by considering heuristics. To find an efficient path, A* orders neighboring vertices it will explore by their perceived cost, which is a combination of the current cost and an estimated cost to the goal.

$$f(i) = \underbrace{g(i)}_{\text{Current Cost}} + \underbrace{h(i)}_{\text{Estimated Cost to Goal}} \tag{3}$$

The major downside of A* is its computational efficiency and long runtimes. When working with vehicles operating at high speeds, we need algorithms that can quickly update as the environment changes around it.

In this project, we utilize a probabilistic method for our local planner. These generate paths by sampling and are useful when it may be difficult to describe the free configuration space but is easy to determine if a configuration is in collision. In its most basic form, the Rapidly-Exploring Random Tree (RRT) algorithm uniformly samples points until it produces a point in the free space. Next, it searches for the nearest existing node on the tree and tries to generate a connection between these two nodes. If the path between these two nodes is collision free, the new node is added to the tree. This process is repeated until a tree is generated which connects the start point to the goal. There are various ways to improve the quality of the path of the algorithm with the cost of increased computational complexity (Algorithm 1).

RRT* adds an optimization layer to the algorithm, leading to more direct and efficient paths. It assigns each node a cost, which we define as the total distance from the current node, through its parent path, to the root of the tree. Instead of only trying to form a connection with the closest node in the tree, RRT* searches for collision free connections within a certain neighborhood, defined geometrically by a distance parameter. It then chooses to connect to the node which reduces its overall cost (Algorithm 2).

Figure 3 shows how RRT* utilizes a generated tree of nodes to construct and efficient path from the start to the goal.

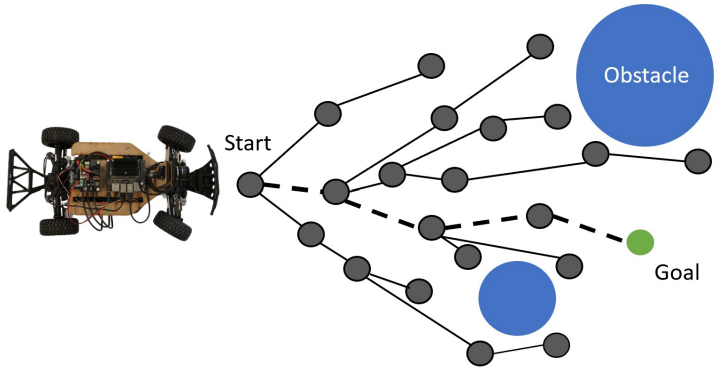
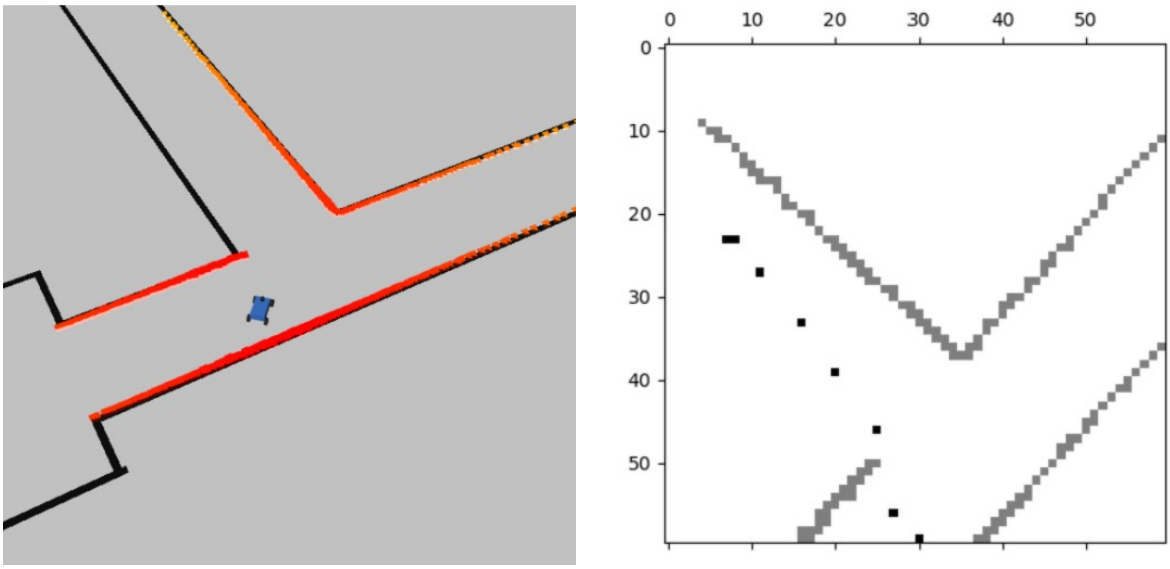


Figure 3: Diagram of RRT* Planner

When planning a path for driving, we seek to reduce the number of turns required such that the vehicle reaches the goal in the most direct fashion possible. For these reasons, the added computational cost of RRT* is well worth the improved results and was thus chosen as the base algorithm for our path planning. Figure 4 shows a side-by-side view of our car in its virtual environment (4a) and the path generated by RRT* (4b) to navigate around the corner and continue along the hallway. The grey areas indicate obstacles and the black points represent the final accepted path from start to goal.



(a) Car in Virtual Environment

(b) Visualization of Planned Path Using RRT*

Figure 4: Visualization of Path Planner

In autonomous driving applications, there are two layers of planners. The planning process we have discussed above is considered a *local planner*. This planner is used to dynamically update short term paths to avoid dynamic obstacles in the vehicles changing environment. For example, if you are driving

down the street and there is a delivery truck stopped in the right lane, you need to adjust your path to enter the left lane and avoid a collision. The second, higher level planner is know as a *global planner*. This planner requires apriori global knowledge such as a map. Commonly, an algorithm such a Dijkstra's is run on a more abstract scale of the world to generate a set of waypoints that can be used as goals for the local planner. For example, if you wanted to drive from West Philly to Center City, your global planner would first decide you must take Chestnut Street before turning on 16th. For the purposes of our demonstration, we circumvented the need for a global planner by manually placing goal points. In our case, we told our car that we want to travel in a loop around Levine hallway. This manually specified set of waypoints worked just as if we had run a global planner with a map and GPS information. Our local planner then determined the exact path through each hallway and around each corner so that we did not collide with any obstacles.

IV.II.II Path Tracking Controller

The purpose of the controller algorithm is to take as input the planned path generated in physical space, and to output motor controller commands such that the path is closely followed. A common approach is to use a proportional–integral–derivative (PID) controller. This algorithm generates a signal command, $u(t)$, based on the time derivative, time integral, and current value of some error function as follows.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{4}$$

In this setting, the error function, $e(t)$, would be a geometric measure of the deviation between the planned path and the current position of the vehicle. The proportional term in the expression is responsible for the theoretically exponential convergence of the vehicle position to the planned path. However, in the realistic case where there is a delay between error measurement and motor command, a purely proportional controller can induce rapid oscillations and possibly overshoots which is undesirable from an efficiency and safety perspective. The derivative term acts to suppress these oscillations by tuning the signal command based on the rate of change of the error, and the integral term acts to correct persisting bias in the error. In theory, PID controllers are robust and can be tuned to achieve desired performance in terms of path smoothness and deviation limits.

One limitation of the PID controller in this setting is that it is not tailored to the application of driving, [1] in particular steering. An ideal controller should be such that it closely follows the planned path while minimizing the number of turns and incorporates the Ackermann steering geometry of the vehicle, something the PID controller does not do. To this end, a more sophisticated tracking algorithm is Pure Pursuit. This algorithm has a tuning parameter referred to as the "look ahead" distance, L. At each iteration of the algorithm, a point which is L units away from the vehicle and intersects the planned path is navigated to via a constant steering angle, γ, computed according to the equation below. The radius of curvature, r, and horizontal distance y are defined according to figure 5.

$$\gamma = \frac{1}{r} = \frac{2|y|}{L^2}$$

[1] Here we are referring to higher-level controllers. On the bottom level, our Vesc motor controller itself implements a version of PID in order to follow the commands supplied by our Pure Pursuit method.

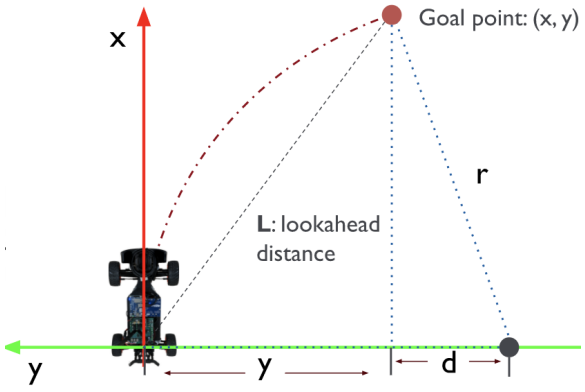


Figure 5: Diagram of Pure Pursuit

The choice of the tuning parameter impacts the degree to which larger errors are tolerated in the interest of a smoother path.

The Pure Pursuit algorithm is advantageous for the present application as the commanded path to be followed is kino-dynamically feasible by construction and possesses low level stability. Additionally, it is generally argued in the autonomy literature that Pure Pursuit is an ideal pairing with RRT or RRT* path planning algorithms as it is possible for the path planner to predict the Pure Pursuit path when evaluating candidate paths for collisions.

IV.II.III Path Prioritization

One of the primary challenges when communicating between the path planner and the path tracker is that the two modules run on different time-scales. That is, a path is published asynchronously whenever the RRT* algorithm converges, while the Pure-Pursuit controller runs at the frequency that it receives localization data. The challenge with merging the path planning and tracking is that potentially contradictory paths can be published in rapid succession relative to the speed at which the vehicle moves, causing the controller to become unstable or susceptible to jitter. To resolve this issue, additional filtering logic was added to the path planner such that for every new path generated by the RRT* algorithm, the previously published path is only overwritten if it now contains collisions in the local frame of the vehicle at the current time. This logic effectively allows us to commit a planned path so long as it is still valid even as the vehicle updates its understanding of its surroundings.

IV.II.IV Current State

Given a 2D map, our RRT* path planner can generate an obstacle free path from the car's current location to a goal point within a few seconds (depending on the size of the local frame). Every time it receives a new map from a ROS topic, it updates the goal point and re-plans a new path to adjust for any obstacles. Once the path is generated, it is published via ROS to the controller. Our pure-pursuit controller is constantly listening for the latest path. Given the cars current position and the latest planned path, the controller generates a series of speed and steering commands. These commands can either be sent to our simulator or directly to the vehicle itself, causing the vehicle to move in the desired fashion. Due to computational constraints[2], we were unable to run SLAM along with all our algorithms

[2]We were unable to acquire the NVIDIA Xavier

simultaneously. On demo day, we verified how with a pre-generated map and an estimated localization (based on forward kinematics), our vehicle could plan and traverse an obstacle free path through its environment. We had a success rate of around 70% and the failures were primarily due to the localization estimation step, which would be eliminated through the pairing with SLAM [3].

IV.III Integration

IV.III.I Specifications

The integration portion of the project consisted of two main sections: connecting the output of perception to the planner and prototyping the entire system on an F1Tenth prototype. In terms of specifications, the car and its accompanying software tools had to be computationally and energy efficient so that it can run as a standalone system in real-time and it must, to a high accuracy, avoid collisions to ensure safety. Importantly, a core requirement of the car is autonomy. To that end, so we built a pipeline that starts with a data stream of events from the event-camera and ends with the physical wheel movements of the car.

IV.III.II Technical Description and Iterations

For the physical F1Tenth car, we used the bill of materials and instructions found at https://f1tenth.org/. The major modifications came as a result of using a DVS instead of a LIDAR as specified in the build instructions. This meant that we had to laser cut a custom platform to accommodate our relevant periphery devices. The remainder of the build involved mounting our VESC motor controller and the Jetson Nano, wiring components to a powerboard, and laser cutting a mount for the camera. Beyond the physical build, our first iteration was to get wheels spinning through by connecting a laptop into the motor controller. This step involved flashing new firmware that enabled turning and the calibrating the PID values.

Once the motor controller could turn the wheels reliably, the next iteration step was to control the car using a joystick over Bluetooth using ROS. To do this, we installed a ROS VESC driver the publishes steering angles to the servo and speeds in RPM to the motor controller. Once we connected the controller, we map joystick movements to ROS topic outputs to drive the car remotely. Our philosophy was that we could replace the joystick input with a python script that publishes to the same topic in order to transition to autonomous driving. This step also involved establishing the autonomy of the car by now fully powering the compute unit with the onboard NiMH battery as well as using Bluetooth for communication.

The final step of the integration was to replace the joystick input with controller output that were generated after path planning. To do so, we wrote a ROS driver python script that read in events from the camera and published them to a ROS topic. Similarly, we used the ROS-RealSense library to publish RGB image data from the Intel RealSense camera to a ROS topic as well. We then modified the ORBSLAM3 and computer vision algorithms to be subscribers to the respective ROS topics in order to read in image/event data and produce a 3D occupancy grid of the environment. We then process this map using Bresenham's line algorithm as described in section **IV.I.I** and publish a 2D map to an intermediate ROS topic. Then, our RRT path planner subscribes to this topic and generates a path given the detected obstacles in the local frame using the methodology explained in section **IV.II.I**. Lastly, the path is communicated via ROS to the controller which publishes speed and turning angles to the motor

[3] Our car had to estimate its position using its theoretical initial position (which due to human error is never where the car actually began) and a state estimator which predicted the current position of the car given the previous motor commands.

controller which drives the car. The new events that arrive in between cycles are buffered until the next iteration of SLAM and optical flow and the entire process is repeated.

IV.III.III Current State

The current state of our car is that, while the entire pipeline is built out, both the perception and planning subsystems cannot be run in parallel due to real time computing constraints imposed by the Jetson Nano. Thus, we segment our prototype functionality into joystick control where we run the perception and mapping to generate a 2D map of the surroundings, followed by running the planner and controller on that map to drive the car autonomously. We make use of wireless telemetry to provide visuals of live optical flow, SLAM, and path generation in order to verify correct operation at each stage of the pipeline.

IV.IV Conclusion

All of our constraints and solutions come together in the construction of the autonomous F1Tenth prototype. Specifically, the car, the hardware periphery and the software must be computationally and power efficient so that it can be run as a standalone system in real-time and it must, to a high accuracy, avoid collisions to ensure safety. Thus, many of the constraints that applied when building both the Perception and Mapping systems as well as the Planning and Controls pipeline apply to the integration of both systems in the car. In the end, by using a visual mapping algorithm in ORBSLAM and cutting out the need for the LIDAR, we aimed to develop a lower cost and more power efficient system for autonomous driving. In the end, even once both of the separate part of the project had been implemented, there was a significant hurdle in integrating everything into a cohesive system. To address that issue, we build in testing capabilities that allowed us to test each system in parallel. For the perception and mapping, we used a collection of our own recorded and online ROSBAGs to test the SLAM map generation, culminating in a live mapping of Tangen Hall using the joystick control. For the planning and control pipeline, we stress tested the planner by adding artificial noise to the ORBSLAM map and simulate the noisiness of a real time generated map. In summary, the main rationale behind the development of event based autonomous systems is that they save on size, weight, and power and, thus, we intend to both write our algorithms and build our car with those constraints in mind.

V Business Overview

V.I Need and Value Proposition

We plan to boil down our project into a business offering, eyeRIS. This is a system-on-a-chip solution consisting of a event-based camera agnostic chip that smoothly integrates into other vehicles' systems and is loaded with our proprietary event-based algorithms, including SLAM, object detection, and optical flow. This product would provide easy access for traditional OEMs, AV companies, and defense companies to an orthogonal dimensionality of data that only event-based vision can capture and cover edge cases like fast turns and dark environments which current solutions fail in. Additionally, the plug-and-play nature of the product allows for rapid integration of event-based vision into existing AV companies' technology stacks. This platform will rapidly accelerate the development cycle of growing AV companies and established defense contractors as they seek to build their sensor suite to be more robust to changing environments and reduce size, weight, and power (SWAP).

V.II Stakeholders

We aim to sell this product to two main segments that would benefit from this: OEMs and defense companies. The former includes both traditional OEMs such as Ford, Toyota, Volvo, etc. (many of whom have begun developing their own AV capabilities) and AV companies like Tesla, Waymo, Cruise, and Aurora. The latter includes companies like Raytheon, Northrup Grumman, and Lockheed Martin. Other stakeholders are complementors (manufacturers of the event-based camera and event-based camera software/firmware vendors), regulators (NHTSA and DoT), and the general public that would use autonomous vehicles with our software.

V.III Market Opportunity & Customer Segments

The current autonomous vehicle market consists of companies operating at various levels of the technology stack. Companies like Tesla and Nuro build their own cars, hardware, and software in house, while Cruise, Aurora, and Argo develop a customized software and sensor suite for autonomous driving, and partner with traditional vehicle manufacturers such as Volvo, Ford, and PACCAR to test their driver systems. Additionally, companies like Mobileye partner with car and defense companies to sell customized hardware and software solutions like lane or collision avoidance systems to them.

We aim to replicate the Mobileye model for the event-based paradigm. Currently, the market for event-based cameras is primarily research-focused and companies don't exist yet that provide these customized hardware and software solutions. Additionally, even the best AV systems haven't reached the required safety levels set by NHTSA to be approved for level 5 autonomy. Our sensor and software suite aims to bridge this gap by covering edge cases where existing camera and LIDAR systems fail, such as low light situations and high speed turns.

We aim to target two main market segments: (1) traditional OEMs, and (2) defense companies. For the traditional OEMs that haven't yet developed any event-based vision specific software, their goal is to improve safety while keeping development costs and time low, so our out-of-the-box solution is perfect as the event-based camera helps improve safety in edge cases. For defense companies, their main goal is to reduce size, weight, and power (SWAP) while maintaining precision, so our event-based solution is the best to achieve both of those objectives.

We plan to partner closely with OEMs and defense companies and develop custom software and hardware acceleration solutions for event-based cameras to work closely with their existing sensor suite and software stack. We will source event based cameras from vendors such as CenturyArks and layer on our hardware acceleration and software suite.

V.IV Market Sizing

The autonomous vehicle market is growing extremely rapidly, with the expected number of level 1 autonomous cars in circulation in 2024 at 54.2 million. Our serviceable addressable market is the global automotive sensors market, which was valued at 24.3 million dollars in 2020, and has a 13.6% expected CAGR through 2026 - the former comes from an estimation that about 2000 dollars are currently spent on the sensor suite of each car.

V.V Competition

As far as we know, there are no direct competitors building a comparable software suite on the event-based paradigm, as it is a relatively new and research-oriented technology. Additionally, with the

current state of progress, we view this product as a supplement to the existing RGB cameras and LIDAR systems rather than a replacement. Rather, the main threat of susbtitution comes from alternative sensors, such as thermal, radar, LIDAR, and ultrasonic. Companies that use these sensors include Velodyne LIDAR, Toposens, Metawave, and Adasky; however, these technologies do not provide the full range of features that event-based cameras do and are not all well suited for high speed situations or have support for computer vision algorithms. Additionally, their systems are not designed for minimal integration with existing technology stacks. We also believe that the risk of substition from RGB cameras and LIDAR getting good on their own to fill in the edge cases is low, given that cameras are limited to a certain number of frames per second, while event based data allow a system to react faster.

Given that the space is just emerging, there is a risk of new entrants competing directly with us. To prevent competitors from entering this space, we would patent the IP behind the event-based SLAM, object detection, and optical flow algorithms.

V.VI Cost & Revenue Model

We plan to partner with OEMs and defense companies to build a custom system-on-a-chip for event-based perception and vision that integrates well into their existing stacks. Given that most of our business will be driven by large contracts with big companies, our revenue will be driven by partnerships for a specific product line (such as a line of new cars or missiles). These contracts will be bespoke and the exact terms will be negotiated, but we estimate charging about 20 million dollars as a flat fee to enter one of these contracts. This will cover about 10 million dollars of estimated development costs from our end as we build the necessary integrations into the company's specific tech stack, leaving us a 50 percent gross margin. Additionally, for every chip that we manufacture, we will charge about 500 dollars, which covers 150 dollars of variable manufacturing cost (which is the average cost of manufacturing an SoC at scale) and 50 dollars of estimated support and maintenance cost. For each chip sale, this leaves us with a 60 percent gross margin.

Other costs we expect to face are fixed development and R&D costs that will be largely concentrated in the first few years of the venture as we build and solidify our IP. We will face incremental R&D costs as we personalize and develop the product further every year.

VI Self-Learning

The F1Tenth website and repository has been an invaluable resource for us. Since we are using a similar hobby race car model, we have been able to leverage their simulator along with maps of Penn hallways and integrated offline sensor tools. By completing sections of the F1Tenth labs and reading through parts of the ROS Wiki, we have familiarized ourselves with publishing and subscribing to topics in ROS. This is crucial for communicating between the various hardware (sensors, motors) and software (controllers and algorithms) nodes in our project. Additionally, we have utilized the ROS wiki for learning how to use Catkin Work spaces. In the words of Sid, just because all of the information is there, doesn't make it easy. When we ran into issues, Billy Zheng graciously assisted us. When it came time to modify our Traxxas car, the F1Tenth website resources were incredibly helpful. In particular, the sections discussing how to power the external compute system off of the battery and setting up the VESC motor controller were of great use.

Through MEAM 520, two of us were exposed to basic planning concepts in Robotics, including a probabilistic method known as RRT. We further researched this topic, reading Sampling-based Algorithms for Optimal Motion Planning and Robotic Path Planning: RRT and RRT* to understand

how we can embed optimization layers into the probabilistic method. Last semester we implemented RRT* on a Franka Panda arm in configuration space. We were able to leverage this experience when implementing the algorithm in the workspace for our vehicle. Our advisor Pratik Chaudhari originally suggested we implement a Pure Pursuit controller. During MEAM 513 (ESE 505) and MEAM 348, some of our team members had an introduction to control theory however this was unfamiliar territory. To educate ourselves, we first read the MATLAB documentation on their built in Pure Pursuit function. We then read a paper from the Robotics Institute at CMU, the developers of this algorithm, which explained its origins, functionality, and benefits. Course content from MEAM 501 regarding mechatronic design was helpful when it came to modifying the Traxxas car.

In ESE 650, we learned how to perform LIDAR-based SLAM and how to update an occupancy grid. ESE 546 has taught us how to work with neural networks, specifically how to train networks and turn parameters. We have reached out to Professor Kostas Daniilidis and a PhD student, Kendall Queen, for help working with the DVS. For help building the car, we have looked at online resources and found Bill of Materials from people who have completed previous builds. Professor Chaudhari has also provided us with the contact information of a few PhD students who work with DVS data.

VII Ethical and Professional Responsibilities

When dealing with autonomous vehicles, the greatest concern is safety. In the interest of the safety of team members, spectators, and our environment we first implemented an emergency braking system for our car. The controller finds the time to collision of all surrounding objects and halts the car if the time is below a certain threshold. Additionally, we implemented and emergency stop feature on our vehicle so that we can quickly disconnect power during testing if any problems arise.

When working with cameras and data collection of any sort, one must always consider the associated privacy implications. All data captured with the DVS will occur in an experimental setting, allowing us to control the environment and ensure that nobody is present who does not consent to being recorded. Similarly, a complex ethical concern for autonomous vehicles is the prioritization of obstacle avoidance and rule breaking. Since our car will only operate in this experimental setting, we will never deal with obstacles other than inanimate objects. By taking the proper safety precautions in advance, the issue of prioritization in obstacle avoidance will not be encountered in testing. Moreover, since the primary innovation and contribution of our project is the DVS-based perception and mapping, applications of this project to real world settings will leave the ethical concerns regarding the path planning of autonomous systems unchanged, and hence are likely to be dealt with in the same way they currently are.

VIII Meetings

Our team met with our advisor, Prof. Chaudhari, on a monthly basis and communicated with him over email on a bi-weekly basis. His support has been primarily on the perception and mapping side of the project and guiding our plan for SLAM algorithms using DVS data. In particular, he recommended using ORB-SLAM and RGB images with artificially introduced noise when developing the planning and controls algorithms. Once we realized a pure event-based SLAM would be out of the scope of this project, Prof. Chaudhari gave us the idea to use the DVS data for dynamic collision avoidance. We discussed future uses of the DVS camera, including a optical flow-based motion planning, where the car moves towards areas of zero flow, which is generally the end of the hallways.

On the planning and controls side, Prof. Chaudhari helped us understand the distinction between the

two layers of planners. He also advised us on how to bridge the gap between the layers and translate a higher level path into goal points that can be utilized by a local planner.

We have also met with our M&T advisor, Dr. Vohra. Her primary advice has been with regards to rationalizing the price point for our business offering, eyeRIS. She recommended that we research SOC products utilized by OEMs that have a similar levels of capabilities to ours. Using their current agreement with vehicle manufactures as an example, we could set a competitive price point for our product.

For additional technical support, we consulted Billy Zheng, a PhD student in the GRASP lab and F1Tenth developer. Billy supplied us with a custom F1Tenth board for our vehicle, which allowed us to power the Jetson Nano from the Traxxas battery. Additionally, he helped Phil and Keshav debug some of their ROS issues related to queue size while they were implementing the planning nodes.

IX Proposed Schedule with Milestones

The fall semester primarily consisted of our team defining the scope of the project and acquiring the necessary background by speaking with our advisors, relevant stakeholders, and PhD students with technical experience in the field. Our Spring schedule can be seen below in Table 1.

On the perception and mapping side, we got the DVS operational and were able to stream the events through a ROS topic and reconstruct the images. At the same time, we were working on getting a basic version of SLAM algorithm running with the RGB camera. Our goal at first was to get a fully event-based SLAM working but we realized this would require a complete rework of SLAM from first principles, so we instead set our goals on incorporating dynamic obstacle avoidance. This changed many of our milestones, with the first step being to try setting up an implementation of SVO-SLAM; however, in order to get it working, we needed some sort of depth measurement to build an accurate map. We quickly scrapped our progress with SVO-SLAM and switched tracks to work on ORB-SLAM3, which we were able to get operational in March. Along the way, we ran into issues with the ORB-SLAM build, and the poor compute of the Jetson Nano prevented us from running the compute-heavy algorithm easily. Regardless, we worked around these issues and got an online version working on the Nano. The next step was to convert this into 3D sparse occupancy grid into a 2D map that the planning and controls team could use. This is still in progress as we debug issues with getting it operational in real time.

The Planning and Controls team set early semester goals of implementing and testing all of our algorithms in simulation. We got a head start over winter break and were able to meet this deadline. Next, we transitioned over to the physical vehicle to make the necessary modifications and calibrate all of the components. We ended up needing a few more connectors and cables than we anticipated, so we were slowed down slightly as we waited for them to arrive. Nonetheless, our vehicle was finished and able to be driven by the start of March. We spent the rest of the semester integrating, debugging, and parameter tuning our algorithms as we transitioned from simulation to the physical vehicle. As it became clear that we had to adjust our demo due to computational constraints, we brainstormed how we could display our technology and capabilities without the aid of the perception and mapping side of the pipeline.

X Teamwork

Our team benefited from the fact that four of our five members are roommates. Thus, our team regularly communicated in person and over text throughout the semester, with weekly meetings taking

Task	Vinay	Anish	Neil	Keshav	Phil
Order F1Tenth Parts	12/22/21				
Complete RRT* in Simulation				01/12/22	01/12/22
Complete Pure Pursuit in Simulation				01/12/22	01/12/22
Get DVS Operational	01/12/22		01/12/22		
Modify and Assemble Vehicle			02/20/22	02/20/22	02/20/22
Calibrate Motors and Sensors on Vehicle	02/28/22	02/28/22	02/28/22	02/28/22	02/28/22
Build ORB-SLAM with RGB Camera	03/04/22	03/04/22	03/04/22		
Build DVS-based ORB-SLAM working on simulated data	03/20/22	03/20/22	03/20/22		
Using Planner and Controller to Move Car			03/20/22	03/20/22	03/20/22
Event-based HOTS-style SLAM working on simulated data	03/25/22	03/04/22	03/04/22		
Event-based HOTS-style SLAM working on Live data	03/30/22	03/30/22	03/30/22		
Entire system navigating in unknown environment	04/28/22	04/28/22	04/28/22	04/28/22	04/28/22

Table 1: Proposed Schedule

place each Friday afternoon to discuss upcoming deliverables and recent progress. All members have contributed to overarching project deliverables including meetings, presentations, and reports.

We split the two major technical components of the project among the team members based on individual expertise, with Vinay, Anish, and Neil being responsible for the perception and mapping side, and Keshav and Phil being responsible for the planning and controls. Keshav has focused primarily on the controller workflow in the ROS architecture, including the development of the PID controller for the wall-following simulation, and preliminary research for the Pure-Pursuit controller to be implemented next semester. Phil, with his background in path-planning algorithms from MEAM 520, has focused implementing RRT and RRT* in physical space once given an occupancy grid (map) of the environment. Together with Neil, the three of them modified and calibrated the vehicle. Anish, with his prior experiencing building a LIDAR version of SLAM in ESE 650, focused primarily on building the ORB-SLAM3 pipeline to generate a 3-D sparse occupancy map of the surroundings and convert it into a 2-D map. At the beginning of the semester, Vinay created the Bill of Materials for the new 1/10th scale RC car and ordered parts. He tested SVO-SLAM as a potential candidate SLAM algorithm for the Intel Realsense camera and helped set up the Jetson Nano so that it was a stable platform to install all the necessary libraries and packages. Finally, Vinay helped with the integration of the perception and mapping pipeline to the behavior and motion planning pipeline. Neil was mostly involved in the integration portion of the project both in terms of building he physical car and designing the communication pipeline between steps. On the software side, Neil was in charge of developing the ROS drivers to encapsulate the data as well as implemented the early object detection and optical flow algorithms to work with event based data.

XI Budget and Justification

Our original budget amounted to around $5300 of expenses, accounting for an F1-Tenth Race Car, two DVS's, car hardware modifications, software packages, datasets, and AWS training costs. Initially, we had planned to loan a car from Penn's ESE 615 (Autonomous Racing) class; however, they had no availability. Due to supply change shortages, we were never able to acquire an NVIDIA Xavier so we stuck with a borrowed Jetson. We ended up over budgeting for software costs and our final, simplified budget can be seen below (Table 2). The total cost of our project amounted to just over $4000.

Item	**Source of Funding**	**Approximate Cost ($)**
DVS Cameras and Accessories	ESE	2500
Traxxas Car	ESE	430
Bateries + Charger	ESE	125
Motor Controller + Accessories	ESE	350
Power Board	F1Tenth	50
Compute Module	GRASP	450
Cables and Connectors	ESE	100
Fasteners	ESE / MEAM Labs	40
Misc. Stock	MEAM Labs	50
Total	-	**4095**

Table 2: Final Budget

XII Standards and Compliance

There are four IEEE standards which guided the design of our project. The first standard measures the impact of AI or autonomous systems on well being (IEEE 7010-2020). The primary motivation behind DarkMode is to provide a solution for edge cases encountered by vision based autonomous driving systems. By providing another stream of high quality information to complement cameras in scenarios where they may struggle, we can improve decision making and reduce accidents. Fewer accidents lead to safer driving conditions and general improved well being. Our second guiding standard deals with autonomous robots in their environments (IEEE 1872.2-2021). Since the goal of our project was to create a prototype which could navigate in a foreign environment, it was crucial that we considered best practices in the field. These primarily include goals promoting safety for living agents along with other autonomous systems.

On the technical side, it is vital that our vehicle has testable levels of transparency (IEEE 7001-2021). This allows for objective assessment of our system. Embracing this standard, we took pride in creating visualizations for nearly all of our software, including our path planner. Additionally, we comply with IEEE 1609.0-2019 which specifies protocols for communication systems on board intelligent transportation systems. We use ROS for all of our internal communication and follow proper UDP and Bluetooth conventions when sending or receiving information from external sources.

XIII Progress this Semester

As of last semester, the perception and mapping team had a working version of event-based object detection and optical flow that worked using the live data stream. This semester, the team focused on building and fine-tuning the ORBSLAM-3 algorithm to take RGB input and build a map of the surroundings. Additionally, the team worked on integrating this with the planning and controls team by converting this 3-D sparse occupancy grid into a 2-D map. The team also tested the mapping capability in multiple hallways in Levine and Tangen.

Over winter break, the planning and controls team began developing the path planner (RRT*) and controller (Pure Pursuit) in simulation. By the middle of February, all of the algorithms were debugged and worked given a 2D occupancy grid in simulation. At the end of March, once the car was ready, we began testing on the actual vehicle. There was a considerable amount of parameter tuning that took place in order to translate the algorithms from the simulation environment to the real world.

On the hardware side, we assembled the entire vehicle throughout the course of the Spring semester. In mid-February once all of the components arrived, we began to strip down the Traxxas stock car and make modifications. We installed a new motor controller, power board, battery pack, and all of our on-board compute. Additionally, we designed and manufactured a base and two camera mounts to securely hold all of the hardware. The final vehicle can be seen in Figure 6. Once we had the vehicle assembled, we calibrated the steering and speed through our motor controller.

Figure 6: DM1 - Completed Prototype

The final part of the semester was spent primarily on integration. It was a challenge to get all of our algorithms to run on the Jetson, which runs a bootstrapped version of Ubuntu. In certain cases, like for the DVS software, we had to reach out the company and acquire source code which we could then compile and install ourselves.

XIV Discussion and Conclusions

Currently, we can generate a map of our surroundings and localize the vehicles position using data from a RGB camera and ORB-SLAM3. Leveraging the data from the event based camera, we perform optical flow and object detection. From the sparse occupancy grid generated by SLAM, we collapse the information into a 2D map of obstacles and free space. Utilizing this map, we perform path planning to reach a goal point from the vehicles current position. Our controller then sends motor commands to the

vehicle so that it follows the generated path. We upgraded a Traxxas RC car with new hardware and a computational suite that has served as our testing platform.

One of the big struggles working with event-based data was successfully adapting traditional perception and mapping algorithms to work with the new paradigm. In many cases, the algorithm needs to be reworked from the ground up. To gain the benefits of DVS data, it is important to maintain its asynchronous nature. Attempting to bucket the DVS data, which was our initial plan, we lose the frequency and asynchronous advantages. To capture the benefits, we would need to use a neuromorphic version of SLAM, where the map is modeled as a gradient map. While we were able to show a proof of concept working with object detection and optical flow on event-based data, we struggled to successfully implement the paradigm with SLAM and mapping in general.

The primary challenges we faced during the spring semester were related to integrating the perception pipeline with the path planning and tracking workflow. One key difficulty was sending information encoded as a map between teams. SLAM generates a 3D sparse occupancy grid composed of singular key points it recognizes across frames. Our planning algorithm requires a 2D map (discretized as a grid) with all boundaries and objects classified as occupied in order to perform properly and at optimal efficiency. Simply flattening the 3D spare grid is not sufficient. After some research and thought, we were able to utilize the ray tracing method described in section IV.I.I. A second key challenge that addressed during testing is that the path planning and tracking algorithms both contain several tuning parameters. The tuned values in the ROS simulator were not optimal for the actual F1Tenth vehicle. Rather, we performed testing to determine the algorithm parameters that correspond to smooth driving of the autonomous vehicle. One parameter that was particularly difficult to tune was the size of our local occupancy grid. When limited to onboard compute, the computational cost of a larger occupancy grid led to longer planning times if the grid was too large. This meant the vehicle would need to stop and would not smoothly reach its goal point.

A key learning for our team has been that we should take advantage of the collective knowledge and expertise of the faculty and students at Penn. We initially approached thinking that Prof. Chaudhari would be our sole source of technical advice, but quickly found that PhD students including Kendall Queen and Billy Zheng could provide unique perspectives as they have tackled similar problems with the ROS environment and F1Tenth system. We also found that the timeline for hardware related components and tasks is often limited by delivery schedule and troubleshooting early on in the testing process.

References

[1] F1TENTH Website and Github Repository.

[2] ROS Wiki and Help Documentation.

[3] Event Cameras: Opportunities and the Road Ahead (CVPR 2020), Jun 2020.

[4] National Highway Traffic Safety Administration. Preliminary Statement of Policy Concerning Automated Vehicles. 2016.

[5] Tim Chin. Robotic Path Planning: RRT and RRT*, Feb 2019.

[6] R. Craig Conlter. Implementation of the Pure Pursuit Path Tracking Algorithm, Jan 1992.

[7] S. Karaman and E. Frazzoli. Sampling-Based Algorithms for Optimal Motion Planning. *2012 American Control Conference (ACC)*, 2012.

[8] MathWorks. Pure Pursuit Controller - MATLAB.

[9] Jun Nagata, Yusuke Sekikawa, and Yoshimitsu Aoki. Optical Flow Estimation by Matching Time Surface with Event-Based Cameras. *Sensors*, 2021.

[10] Etienne Perot, Pierre de Tournemire, Davide Nitti, Jonathan Masci, and Amos Sironi. Learning to Detect Objects with a 1 Megapixel Event Camera. *Neural Information Processing Systems*, 2020.

[11] Antoni Rosinol Vidal, Henri Rebecq, Timo Horstschaefer, and Davide Scaramuzza. Ultimate SLAM? Combining Events, Images, and IMU for Robust Visual SLAM in HDR and High-Speed Scenarios. *IEEE Robotics and Automation Letters*, 3(2):994–1001, 2018.

Appendix

Planner Algorithms

Algorithm 1 RRT Path Planner

$\mathrm{T}_{\text{start}} \leftarrow (q_{\text{start}}, \varnothing)$
$\mathrm{T}_{\text{goal}} \leftarrow (q_{\text{goal}}, \varnothing)$
while $n <$ maxSteps **do**
    $q_{\text{new}} = q_{\text{rand}} \in Q_{\text{free}}$
    $q_a =$ Closest Node to q_{new} in $\mathrm{T}_{\text{start}}$
    if NOT collide(q_{new}, q_a) **then**
        add (q_{new}, q_a) to $\mathrm{T}_{\text{start}}$
    $q_b =$ Closest Node to q_{new} in T_{goal}
    if NOT collide(q_{new}, q_b) **then**
        add (q_{new}, q_b) to T_{goal}
    if q_{new} connected to $\mathrm{T}_{\text{start}}$ and T_{goal} **then**
        break

Algorithm 2 RRT* Path Planner

$T_{start} \leftarrow (q_{start}, \varnothing, 0)$
$T_{goal} \leftarrow (q_{goal}, \varnothing, 0)$
while $n <$ maxSteps **do**
    $q_{new} = q_{rand} \in Q_{free}$
    $q_a =$ Closest Node to q_{new} in T_{start}
    $Q_a =$ Nodes in Neighborhood of q_{new} in T_{start}
    if NOT collide(q_{new}, q_a) **then**
        add (q_{new}, q_a, cost(q_{new}, q_a)) to T_{start}
        for node in Q_a **do**
            if cost(q_{new}, node) $<$ cost(q_{new}, q_{new}.getParent()) and NOT collide(q_{new}, node) **then**
                update q_{new} in T_{start} to q_{new}, node, cost(q_{new}, node))
    $q_b =$ Closest Node to q_{new} in T_{goal}
    $Q_b =$ Nodes in Neighborhood of q_{new} in T_{goal}
    if NOT collide(q_{new}, q_b) **then**
        add (q_{new}, q_b, cost(q_{new}, q_b)) to T_{goal}
        for node in Q_b **do**
            if cost(q_{new}, node) $<$ cost(q_{new}, q_{new}.getParent()) and NOT collide(q_{new}, node) **then**
                update q_{new} in T_{goal} to (q_{new}, node, cost(q_{new}, node))
    if q_{new} connected to T_{start} and T_{goal} **then**
        break