# Dynamic Object Avoidance (DOA)
*CPEN 391 Group 5 Project Report*

ASHKAN AHMADY AFKHAM, GEOFFREY BIAN, JORDAN CHEN, LAKSHYA MALHOTRA, EVAN NAWFAL, and JEFFREY YU

## 1 Abstract

We developed a Dynamic Object Avoidance (DOA) system for the F1TENTH autonomous racing car. The system detects, tracks, predicts trajectories of, and safely avoids small, fast-moving dynamic obstacles such as tennis balls. We implemented a modular ROS2 architecture consisting of five interconnected nodes: object detection using an RGB-D camera with HSV segmentation, state estimation with an Extended Kalman Filter, trajectory prediction with Time-to-Collision computation, collision assessment, and path planning with predictive masking integrated into a gap-following controller. The system demonstrates predictive avoidance capabilities, anticipating collisions rather than simply reacting to current sensor data. Key technical contributions include camera-based tracking of small objects difficult for LiDAR to perceive, a predictive masking strategy that injects forecasted trajectories into LiDAR scans as virtual obstacles, and a clean modular architecture enabling independent development and testing of components.

Fig. 1. Dynamic obstacles (deer) in real-life driving.

## 2 Literature Review

### 2.1 Industry Solutions

Autonomous vehicles from Waymo and Tesla implement systems, which handle dynamic objects including moving traffic, road users, animals, and debris. Their implementation utilizes LiDAR or camera sensors (similar to ours) and predicts with advanced algorithms based on Model Predictive Control and Deep Reinforcement Learning. The vehicle then selects the safest, most comfortable, and most efficient collision-free path.

### 2.2 F1TENTH Solutions

Existing F1TENTH solutions solve for static obstacle avoidance such as Gap Following and Wall Following. Some F1TENTH solutions track other dynamically moving vehicles in race settings. However, these models typically neglect detailed perception and prediction of small, fast-moving

Authors' Contact Information: Ashkan Ahmady Afkham; Geoffrey Bian; Jordan Chen; Lakshya Malhotra; Evan Nawfal; Jeffrey Yu.

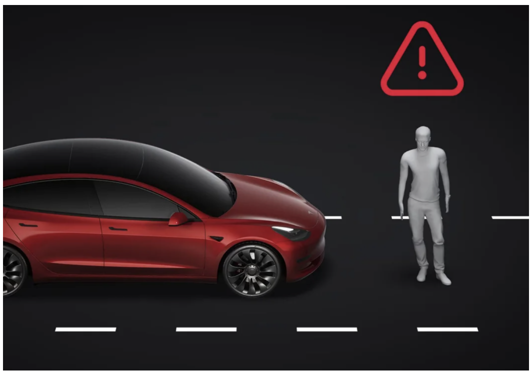


Fig. 2. Camera detection of moving pedestrian by Tesla Motors.

objects, which our system addresses through camera-based detection and predictive trajectory estimation.

## 3 Methodology

### 3.1 Project Method

The system uses a three-stage, modular approach for Dynamic Object Avoidance (DOA) via a publish/subscribe communication model:

(1) Object Detection: This module publishes the current Object Location (position) of dynamic obstacles.
(2) Trajectory Prediction: This module subscribes to the location data, calculates the object's probable future path, and then publishes the resulting Object Trajectory.
(3) Maneuver: This final module subscribes to the predicted trajectory and generates the necessary avoidance path and control commands (steering/braking) to safely navigate around the obstacle.

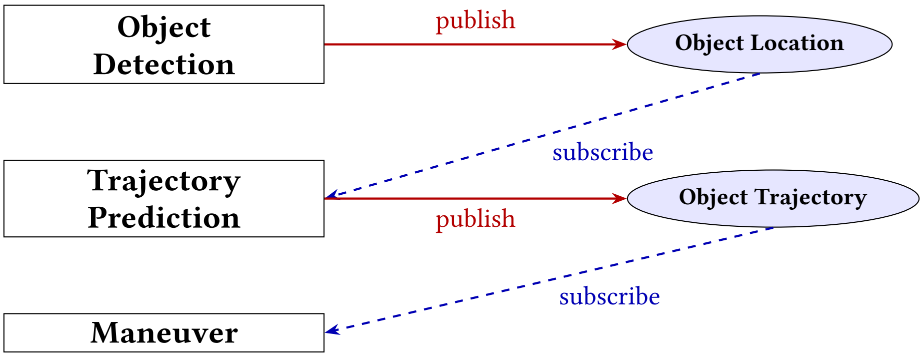


Fig. 3. Data Flow for Dynamic Object Avoidance

## 4 Implementation

The Dynamic Object Avoidance (DOA) system was implemented as three distinct ROS2 nodes, communicating asynchronously via standard publish/subscribe topics. This modular architecture, illustrated in Figure 3 in the Methodology, ensures that the sensor processing, prediction, and path planning steps can execute in parallel, minimizing overall system latency.

| Module | Input Topic(s) | Core Algorithm | Output Topic |
|---|---|---|---|
| **1. Object Detection** | `/camera/color/image_raw`, `../depth/image_rect_raw`, `../color/camera_info` | Vision Processing, Pinhole Projection | `/doa/object_3d_position` ($\mathbf{P}_k$) |
| **2. Trajectory Prediction** | `/doa/object_3d_position`, `/tf` (for Frame Transform) | 4-State Constant Velocity KF | `/tracks/estimate` ($\mathbf{P}, \mathbf{V}$) |
| **3. Maneuver** | `/scan` (LiDAR), `/tracks/estimate`, `/brake` | Gap Following, Dynamic Masking, PD Control | `/drive` (Steering, Speed) |

Table 1. Dynamic Object Avoidance (DOA) System Communication and Architecture

## 4.1 Object Detection and State Initialization

The initial challenge lies in accurately detecting small, high-speed objects (such as tennis balls), which have a minimal footprint on LiDAR and lack robust feature sets for standard visual trakers. This module leverages the Intel RealSense Camera and the object's specific color signature (bright yellow).

*4.1.1 Localization.* The vision pipeline uses synchronized color and depth data to produce a raw 3D position measurement:

- **Camera Synchronization**: The node subscribes to the camera's intrinsic parameters $(F_X, F_Y, C_X, C_Y)$ via `/camera/color/camera_info`. A ROS2 ApproximateTimeSynchronizer is used on the color and depth topics to ensure precise pairing of sensor data.
- **Color Segmentation**: The color image is converted to the HSV color space. A predefined static color range creates a binary mask, isolating the target object.
- **Contour and Depth Processing**: The largest contour is identified. The centroid $(u, v)$ is calculated using image moments. To ensure a robust depth measurement, a square depth window centered at $(u, v)$ is sampled, and the median depth $Z_{\text{cam}}$ is selected to filter outliers.
- **3D Projection and Output**: The final 3D coordinates are obtained via the pinhole camera model projection:

$$Z_{\text{cam}} = \frac{\text{depth}_{\text{mm}}}{1000}, \quad X_{\text{cam}} = Z_{\text{cam}} \frac{u - C_X}{F_X}, \quad Y_{\text{cam}} = Z_{\text{cam}} \frac{v - C_Y}{F_Y}$$

These coordinates are then transformed into the vehicle's base link coordinate frame (X-forward, Y-left, Z-up) and published as the raw Object Location ($\mathbf{P}_k$) on the `/doa/object_3d_position` topic.

*4.1.2 Raw Velocity Estimation.* From the stream of raw positions, the instantaneous relative velocity $\mathbf{v}_{\text{raw}}$ is derived by observing the change in coordinates $\Delta\mathbf{P} = \mathbf{P}_k - \mathbf{P}_{k-1}$ between consecutive time steps $\Delta t$:

$$\mathbf{v}_{\text{raw}} \approx \frac{\Delta \mathbf{P}}{\Delta t}$$

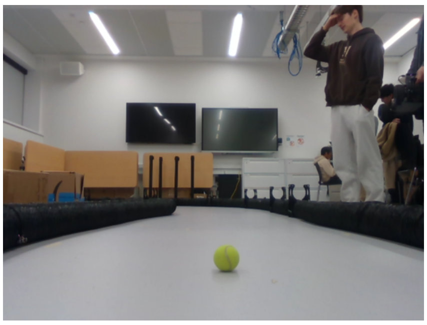


Fig. 4. Tennis Ball on Track

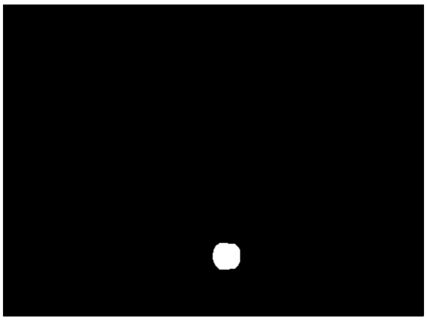


Fig. 5. Tennis Ball Masked

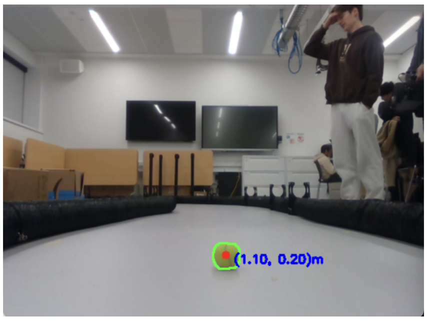


Fig. 6. Tennis Ball Projected

### 4.2 Trajectory Prediction and Filtering

This module produces a stable, low-noise state estimate of the object (position and velocity) for the prediction pipeline. We utilize a 4-state Constant Velocity (CV) Kalman Filter (KF), which proved sufficiently accurate and computationally lightweight for the F1TENTH platform.

*4.2.1 The 4-State Constant Velocity Model.* The filter maintains a state vector $\mathbf{x}_k$ defined by the object's position and velocity in 2D:

$$\mathbf{x}_k = \begin{bmatrix} p_x \\ p_y \\ v_x \\ v_y \end{bmatrix}$$

*4.2.2 Kalman Filter Steps.*

(1) **Prediction**: The state is projected forward based on the time step $\Delta t$ and the constant velocity assumption. The State Transition Matrix $\mathbf{F}$ is:

$$\mathbf{F} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The predicted state is $\hat{\mathbf{x}}_k^- = \mathbf{F}\hat{\mathbf{x}}_{k-1}$. The Process Noise Covariance $\mathbf{Q}$ is incorporated to account for unmodeled acceleration.

(2) **Update**: The new raw position measurement $z_k$ is fused with the predicted state $\hat{\mathbf{x}}_k^-$. The Kalman Gain ($\mathbf{K}$) is calculated to balance trust between the prediction and the measurement. The final filtered state $\hat{\mathbf{x}}_k$ is calculated as:

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}(z_k - \mathbf{H}\hat{\mathbf{x}}_k^-)$$

where $\mathbf{H}$ is the Measurement Matrix that maps the 4-state vector to the 2-state position measurement.

The updated state vector $\hat{\mathbf{x}}_k$ is published as the object state for the path planner.

## 4.3 Impact Prediction

A Trajectory Projection Node consumes the 4-state constant velocity estimate $\hat{\mathbf{x}}_k$ published by the Kalman Filter. Its core function is to deterministically forecast the object's future state to calculate the precise point and time of collision.

## 4.4 Maneuver and Dynamic Avoidance

This final module, based on an extension of the Gap Following algorithm, generates the evasive control commands. It intelligently combines LiDAR data with the object's predicted path. The node subscribes to the LiDAR Scan and the Predicted Impact Point, $\mathbf{P}_{\text{impact}} = (x_{\text{predicted}}, y_{\text{predicted}})$ from the trajectory node.

*4.4.1 Path Biasing.* The key innovation is the use of the predicted impact point to bias the path planning away from the collision zone:

(1) **Angle Calculation**: The angle $\theta$ to the predicted impact point in the vehicle's frame is calculated:

$$\theta = \arctan 2(y_{\text{predicted}}, x_{\text{predicted}})$$

(2) **LiDAR Masking**: An angular sector around $\theta$, determined by a safety margin, is calculated. All LiDAR rays within this sector are forcefully set to a range of 0.0 m.

(3) **Path Biasing**: By masking this region, the subsequent Gap Following logic treats the predicted path of the ball as a solid, static obstacle. This compels the algorithm to choose the largest available gap that is safely to the side of the predicted impact zone, successfully steering the vehicle away from the predicted collision.

*4.4.2 Gap Selection and Control.* The modified LiDAR scan is processed by the standard Gap Following steps: applying a safety bubble, disparity extension, and selecting the center of the largest gap.

- **Steering Control**: A PID Controller is used to calculate the steering angle based on the error:

$$\text{Steering Angle} = K_p \cdot \text{Error} + K_d \cdot (\text{Error}_{\text{prev}} - \text{Error})$$

- **Dynamic Speed Adjustment**: The vehicle's speed is dynamically changed based on the magnitude of the required steering angle, prioritizing stability during evasive maneuvers.

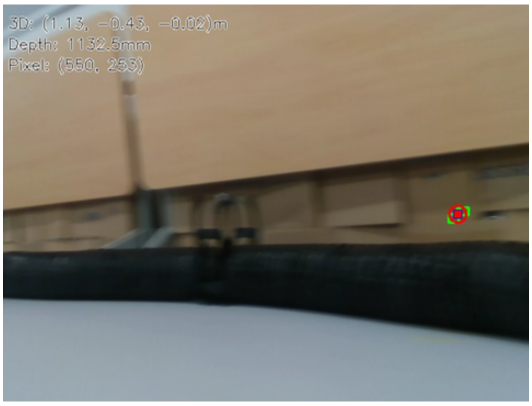
## 5 Challenges

Throughout the implementation of this Dynamic Object Avoidance system, we encountered several technical challenges that required careful debugging and creative solutions.

### 5.1 HSV Color Segmentation Tuning

The object detection module relies on HSV color thresholds to isolate yellow tennis balls from the background. We faced significant difficulties with:

- **Lighting sensitivity:** Finding HSV thresholds that reliably detected yellow tennis balls required extensive testing. We experimented with various ranges and settled on (Hue: 20-40, Saturation: 100-255, Value: 100-255) that balanced detection sensitivity with false positives under lab lighting conditions.
- **Background interference:** Yellow-ish floor tiles and other objects sometimes triggered false detections. We filtered out small noise by requiring detected objects to be at least 100 pixels in area and applying image processing operations to clean up the detection mask.
- **Ball occlusion:** Partial visibility resulted in irregular contours. We used moment-based centroid calculation rather than relying on perfect circular shapes.



```
3D: (1.13, -0.43, -0.02)m
Depth: 1132.5mm
Pixel: (550, 253)
```

Fig. 7. False Reading

### 5.2 Depth Camera Noise and Invalid Readings

The Intel RealSense depth camera produced noisy and unreliable depth measurements, particularly for small objects:

- **Sparse depth data:** Small tennis balls sometimes lacked valid depth pixels. We implemented a windowed approach (5x5 pixel window around the centroid) and used median filtering to extract robust depth estimates.
- **Invalid depth values:** The camera frequently returned zero or infinite depth readings. We added validation logic to ignore invalid depths and prevent crashes.
- **Depth-color misalignment:** Even with hardware synchronization, slight temporal offsets between color and depth frames caused position errors. We used ROS message_filters with ApproximateTimeSynchronizer to align frames within 100ms.

### 5.3 Coordinate Frame Transformations

Converting between camera frame, vehicle frame, and world frame proved more complex than expected:

- **Pinhole camera model:** We had to correctly apply the camera intrinsic parameters (fx, fy, cx, cy) from the CameraInfo topic to project 2D pixel coordinates to 3D space. Initial implementation errors caused objects to appear in incorect positions.
- **Camera-to-vehicle transform:** The camera is mounted forward-facing on the vehicle, requiring axis transformations (camera Z becomes vehicle X, camera -X becomes vehicle Y, camera -Y becomes vehicle Z). Getting these sign conventions correct took several iterations.
- **TF2 transforms:** Integrating with ROS2 TF2 for dynamic coordinate transforms required careful handling of timestamps and exception handling for missing transforms.

### 5.4 Kalman Filter Tuning

The Extended Kalman Filter required significant tuning to balance responsiveness and stability:

- **Process noise covariance:** Setting appropriate process noise (acceleration variance) was critical. Too low caused the filter to ignore measurements; too high made it overly sensitive to noise. We settled on 1.0 $m/s^2$ after empirical testing.
- **Measurement noise covariance:** Camera-based measurements had roughly 5cm standard deviation error. We set measurement noise to $0.05^2$ accordingly.
- **Filter initialization:** The first detection had no velocity information, requiring careful initialization of the state covariance matrix with higher uncertainty in velocity components.
- **Timestamp handling:** Computing accurate time deltas between measurements was essential for proper prediction. We had to handle edge cases like negative or excessively large time steps.

### 5.5 Time-to-Collision Edge Cases

The trajectory prediction node needed robust handling of various geometric scenarios:

- **Diverging objects:** Objects moving away from the vehicle (positive relative velocity) should not trigger avoidance. We added checks to ignore these cases.
- **Near-zero velocities:** When objects were nearly stationary, TTC calculations could produce infinite or undefined values. We implemented minimum velocity thresholds and TTC bounds (0.1s to 10s).
- **Lateral miss distances:** Computing whether the object would actually hit the car required careful geometric analysis of predicted lateral positions relative to vehicle width plus object radius.

### 5.6 ROS2 Integration and Timing

Coordinating multiple nodes with different update rates presented synchronization challenges:

- **Message type selection:** Choosing appropriate ROS message types (PointStamped, Odometry, Float32, Bool) to efficiently convey information between nodes.
- **Debugging distributed systems:** Tracing issues across multiple asynchronous nodes required extensive logging and ROS topic monitoring (ros2 topic echo, rqt_graph).

## 5.7 Real-Time Performance Constraints

Ensuring the system operated fast enough for dynamic obstacle avoidance:

- **Image processing latency:** HSV conversion, morphological operations, and contour detection added computational overhead. We used efficient OpenCV operations to minimize processing time.
- **Control loop frequency:** Ensuring the gap follower received updated predictions frequently enough to react smoothly.

## 5.8 Gap Follower Integration

Integrating predictive masking with the existing gap-following algorithm required careful design.

- **Impact zone masking:** Determining the appropriate angular width (±8 degrees) for masking predicted impact regions in the LiDAR scan to ensure the gap follower avoided them without being overly conservative.
- **Prediction availability:** Handling cases where predictions were unavailable (no object detected) gracefully without disrupting normal gap-following behavior.
- **Coordinate consistency:** Ensuring predicted impact points in the vehicle frame aligned correctly with LiDAR angular coordinates.

## 5.9 Individual Contributions

Each team member contributed to both implementation and testing phases of the project. Table 2 summarizes the primary responsibilities and additional contributions of each member.

| **Team Member** | **Primary Contributions** |
|---|---|
| Geoffrey Bian | Initial object detection node; RGB-D camera integration; HSV color segmentation; camera/detection tuning during testing. |
| Jordan Chen | Camera calibration; detection enhancements; Kalman filter draft; ROS2 setup and launch files; documentation; testing coordination. |
| Jeffrey Yu | Kalman filter completion; world-frame coordinate transforms; full EKF implementation; Kalman filter tuning during testing. |
| Ashkan Afkham | Trajectory prediction node; Time-to-Collision algorithm; collision detection logic; impact point estimation; trajectory/collision testing. |
| Evan Nawfal | Avoidance path planning; gap-follower enhancement with predictive masking; control/avoidance parameter tuning during testing. |
| Lakshya Malhotra | Emergency braking functionality for avoidance path node; ROS2 system setup; planning and design support; demo preparation; software integration testing/debugging |
| **All Members** | Hardware testing; parameter tuning; single and multi-object testing; performance benchmarking. |

Table 2. Team Member Contributions

All implementation work is documented in the project GitHub repository with individual commit histories reflecting the contributions listed above.

## 6 Conclusions

This project successfully implemented a Dynamic Object Avoidance (DOA) system for the F1TENTH autonomous racing platform. Our system performs real-time detection, tracking, trajectory prediction, and collision avoidance for small, fast-moving obstacles like tennis balls.

### 6.1 Summary of Achievements

We developed a modular ROS2 architecture with five interconnected nodes:

1. **Object Detection:** Uses an Intel RealSense RGB-D camera with HSV color segmentation and 3D projection to localize tennis balls.
2. **State Estimation:** An Extended Kalman Filter (EKF) smooths noisy camera measurements to estimate object position and velocity.
3. **Trajectory Prediction:** A constant-velocity motion model forecasts future positions and computes Time-to-Collision (TTC).
4. **Collision Assessment:** Checks if the predicted trajectory intersects the vehicle's path.
5. **Path Planning:** Masks predicted impact zones in the LiDAR scan, forcing the gap-follower to generate safe evasive maneuvers.

The key advancement is predictive avoidance: the vehicle anticipates collisions instead of only reacting to current sensor data.

### 6.2 Technical Contributions

- **Camera-based tracking:** Combines RGB and depth data to detect small objects that LiDAR struggles to perceive.
- **Predictive masking:** Uses the predicted object paths to "paint" virtual obstacles into the LiDAR scan, so the planner can react to them as if they were real.
- **Modular architecture:** Clean separation of detection, tracking, prediction, and planning for easier debugging and enhancement.

### 6.3 Lessons Learned

- Coordinate frame transforms between camera, vehicle, and world frames are critical for accuracy.
- Robust filtering is essential due to sensor noise.
- Synchronizing sensor streams and managing ROS message timing requires careful timestamp handling.
- System performance depends heavily on parameter tuning (HSV thresholds, control gains, safety margins).

### 6.4 Future Work

Several improvements could enhance the system:

- **Deep learning detection:** Use YOLOv8 or similar models for robust multi-object detection in varied lighting.
- **Advanced motion models:** Incorporate physics (friction, bouncing) or train LSTM networks for better prediction.
- **Multi-object tracking:** Implement data association algorithms to handle multiple simultaneous threats.

- **Sensor fusion:** Combine LiDAR and camera for improved depth accuracy and IMU for motion compensation.
- **Optimal planning:** Use Model Predictive Control (MPC) or Rapidly-exploring Random Tree Star (RRT*) for smoother, more efficient trajectories.
- **Learning-based approaches:** Apply reinforcement learning for end-to-end policies or online adaptation.

### 6.5 Closing Remarks

This project demonstrates that dynamic object avoidance is feasible on autonomous platforms using modular sensor fusion and predictive planning. By combining vision, state estimation, and trajectory prediction, we created a system that anticipates threats rather than just reacting to them. The techniques developed here provide a foundation for autonomous vehicles operating in dynamic environments.