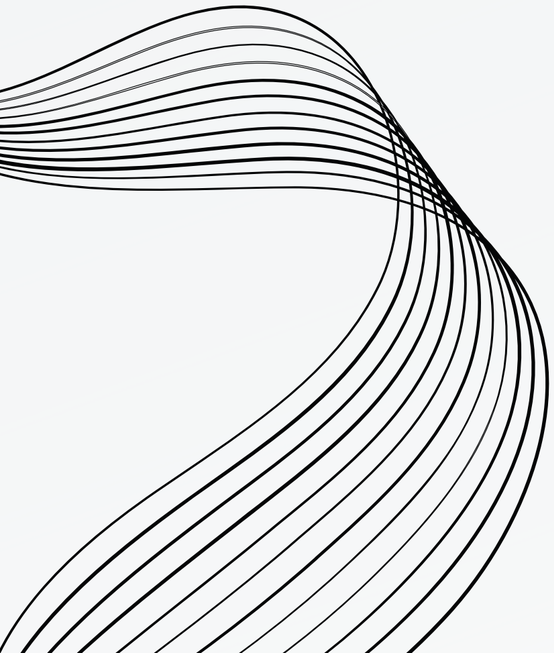
CHAUFFEUR: Benchmark Suite for Design and End-to-End Analysis of Self-Driving Vehicles on Embedded Systems

CS637A - Course Project

Course Instructor: Dr. Indranil Saha



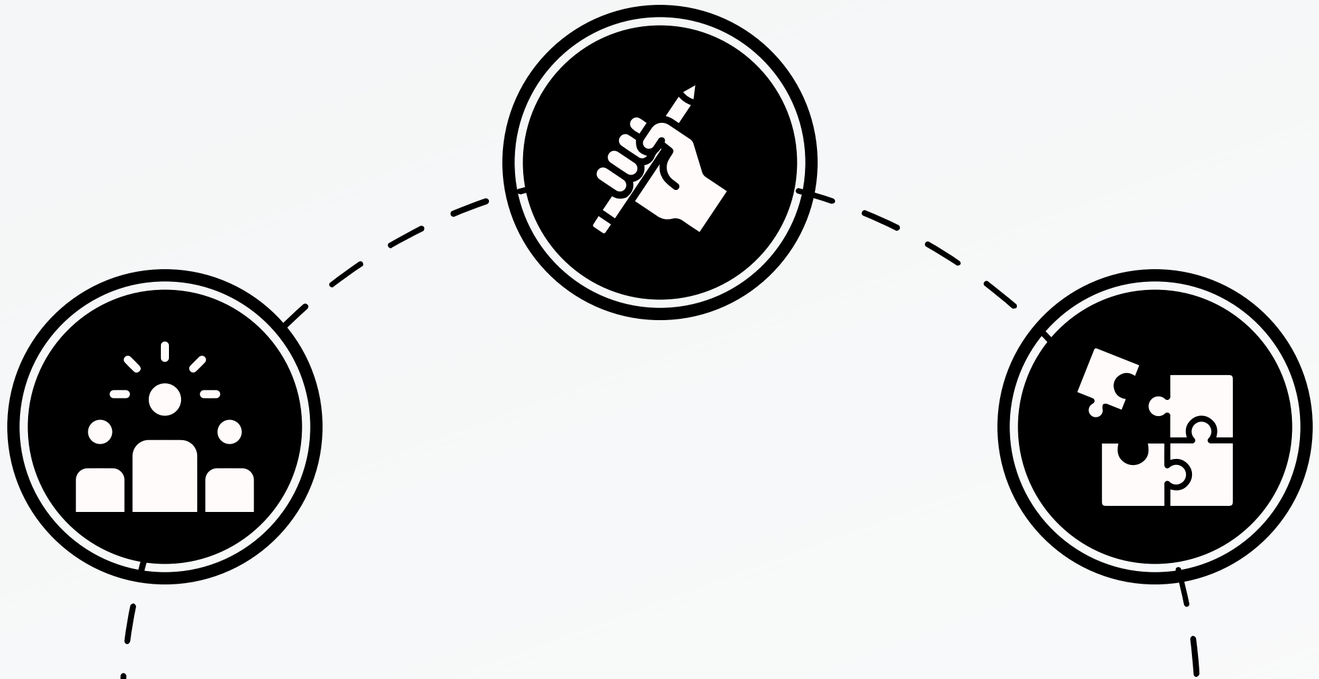
Group 6

Arnav Pandey (200188)
Ritam Jana (200798
Utkarsh Kandi (201068)
Prabuddha Singh (200691)
Aditya Palmate (200662)

NVIDIA®

GOALS AND OBJECTIVES

To introduce "Chauffeur," an open-source benchmark suite for self-driving vehicles. To highlight the motivation behind creating Chauffeur and the challenges it addresses.



LIMITATIONS OF EXISTING BENCHMARK SUITES

Traditional benchmarks:

- Conventional embedded benchmark suites (e.g., MiBench, SPEC, and PARSEC) cover various application fields, such as media processing, computer vision, animation physics, and corporate servers.
- These benchmark suites primarily focus on evaluating system/hardware design and computer architecture.

- Existing benchmarks are often specialized for either GPU workloads (e.g., Rodinia) or CPU workloads (e.g., PARSEC), resulting in a limited scope.
- This limited scope fails to accurately capture the diverse resource utilization ratios that are typical in self-driving cars.

LIMITATIONS OF EXISTING BENCHMARK SUITES

CAVBench:

- CAVBench is a benchmark suite specifically designed for assessing the performance of computing systems in the context of autonomous driving, particularly in a connected vehicle environment.
- It encompasses six key workloads relevant to self-driving cars: Simultaneous Localization and Mapping (SLAM), object detection, object tracking, battery diagnostics, speech recognition, and edge video analysis.

- CAVBench provides an in-depth analysis of the execution time breakdown for each of these applications, offering insights into the computational demands of autonomous driving tasks.
- CAVBench serves as an initial artifact to study edge computing systems for autonomous driving but fails to present a holistic view of the end-to-end self-driving scenario.

LIMITATIONS OF EXISTING BENCHMARK SUITES

Autoware:

- Autoware is a widely used open-source full-stack driving simulator built on the ROS and leverages various established open-source software libraries.
- However, Autoware and similar full-stack simulators often demand powerful hardware platforms for evaluation. For instance, for assessing Autoware includes an 8-core X86 CPU with 32GB of RAM, making it impractical for resource-constrained embedded platforms.

- To study workloads on more specialized hardware like NVIDIA DRIVE PX2 computing platform, researchers have had to make customizations to its software stack. This customization, while enabling specific evaluations, limits the ability to explore different algorithms for the same task.
- The porting of Autoware to emerging embedded platforms involves significant efforts and complexity, posing challenges for hardware designers and researchers.

LIMITATIONS OF EXISTING BENCHMARK SUITES

Apollo:

- Apollo is an industry-standard autonomous driving software framework that is developed with the primary application of self-driving vehicles in mind.
- Operating Apollo and similar frameworks on low-resource embedded hardware can be challenging, as these frameworks are designed with a focus on functionality rather than resource efficiency.

- Setting up an end-to-end driving stack for hardware and architectural studies using frameworks like Apollo is time-consuming and burdensome for those seeking to assess the hardware implications of autonomous driving systems.
- Existing benchmarks are specialized for either GPU workloads (e.g., Rodinia) or CPU Researchers and hardware designers face difficulties in adapting such complex software.

WHY CHAUFFER?

Representation of Real-World Workloads

Chauffeur includes applications that accurately represent the diverse, sensor-driven, and real-time workloads encountered in self-driving systems.

End-to-End Pipeline Analysis

Chauffeur enables the analysis and optimization of the entire self-driving pipeline, allowing researchers to understand the interplay of various stages and components.

Embedded System Focus

Chauffeur addresses the unique challenges of resource-constrained embedded systems, such as real-time performance, energy efficiency, and safety.

WHY CHAUFFER?

Heterogeneity Support

Chauffeur accommodates the trend of utilizing GPUs and dedicated accelerators, ensuring that self-driving applications can make efficient use of available resources.

Diverse Platform Compatibility

Chauffeur is designed to work on a range of embedded platforms, reflecting the industry's use of various hardware configurations.

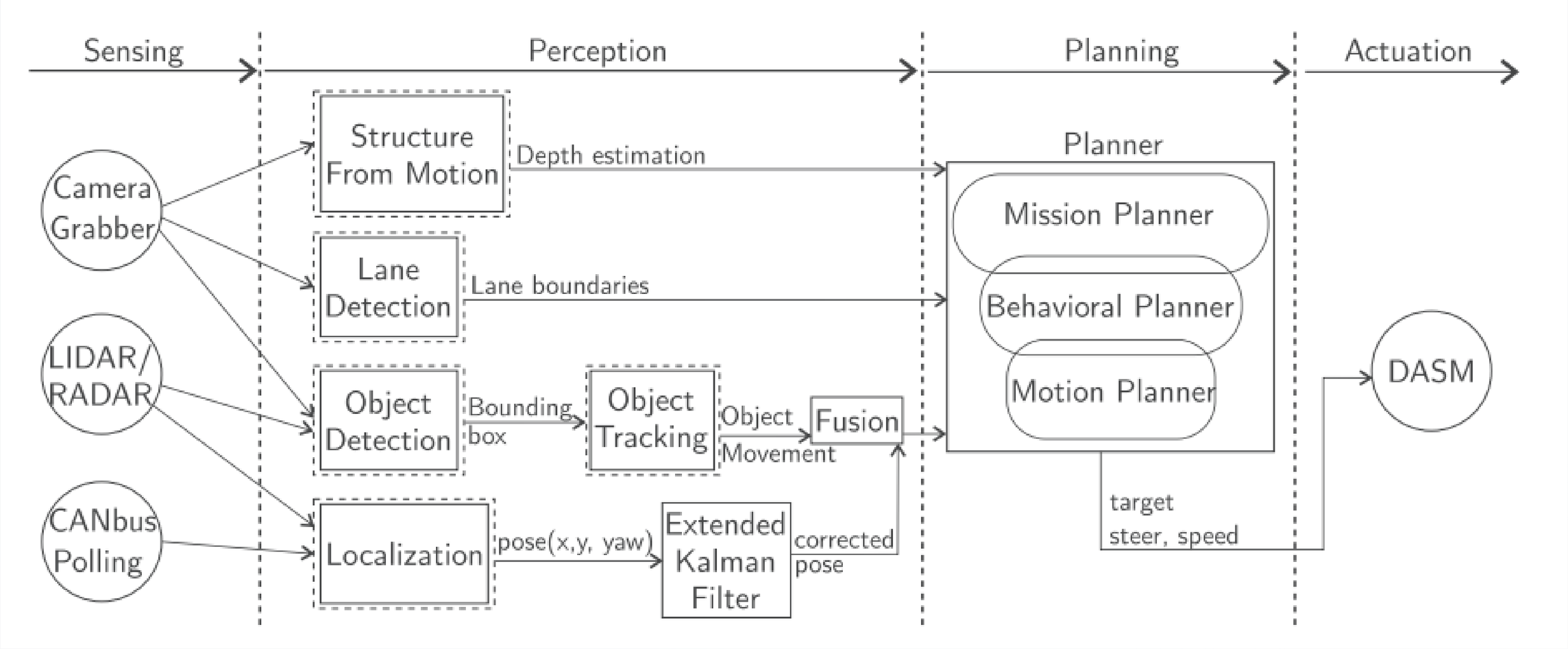
Open-Source and Research-Friendly

Chauffeur is an open-source benchmark suite that makes it easy for researchers, hardware designers, and system developers to evaluate and experiment with self-driving solutions.

COMPARISON OF POPULAR BENCHMARK SUITES

	Traditional benchmarks	CAVBench	Autoware	Apollo	**Chauffeur**
Self Driving Workload	✗	✓	✓	✓	✓
Configurable end to end pipeline	✗	✗	✗	✗	✓
Embedded Runtime	✓	✗	✓	✗	✓
Heterogeneity	✗	✓	✓	✓	✓
Diverse Platforms	✓	✗	✗	✗	✓
Supports research	✓	✓	✓	✓	✓

SELF-DRIVING PIPELINE

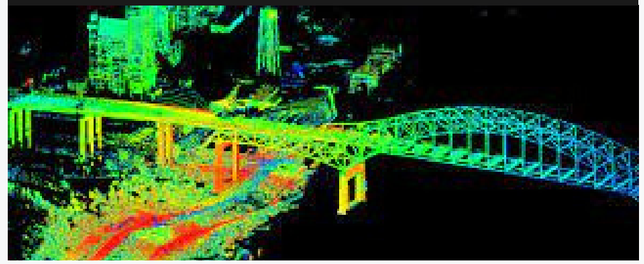


SENSORS

Camera

- Visual sensors are a vital component to enable the perception of the environment in self-driving vehicles.
- Require a high-bandwidth communication bus because they interface multiple cameras simultaneously.
- The camera grabber is responsible for receiving the packets from the network and storing them for later stages.

LIDAR/RADAR



- Detection and ranging sensors are used to detect surrounding objects and calculate distances. Distance sensors assist in hazard detection and range-finding.
- These sensors are imperative during adverse weather and lighting conditions.

CAN Bus Polling

- CAN bus is typically used for low-volume data transfers with high reliability.
- Used for reading Odometer values, interfacing with sensors such as IMU and controlling the steer and speed of the vehicle.

CAN BUS POLLING

- CANbus polling, also known as Controller Area Network (CAN) polling, is a communication method used to collect real-time data from hardware sensors in automotive and industrial applications

- It allows multiple devices, such as sensors, controllers, and actuators, to communicate efficiently and reliably on a shared network.

How does CANbus Polling work?

Sensor Network Setup

•Multiple sensors are connected to a CANbus network

•Each sensor is assigned a unique identifier or address (CAN ID).

•A central controller or monitoring device, such as an Engine Control Unit (ECU) in a vehicle or a Programmable Logic Controller (PLC) in an industrial setting, is also connected to the CANbus network.

Polling Process

•The central controller initiates the polling process by sending a request (CAN message) with the specific CAN ID of the sensor it wants to query.

•The sensor that matches the requested CAN ID responds with its data.

•The central controller receives and processes the data.

Real Time Data collection

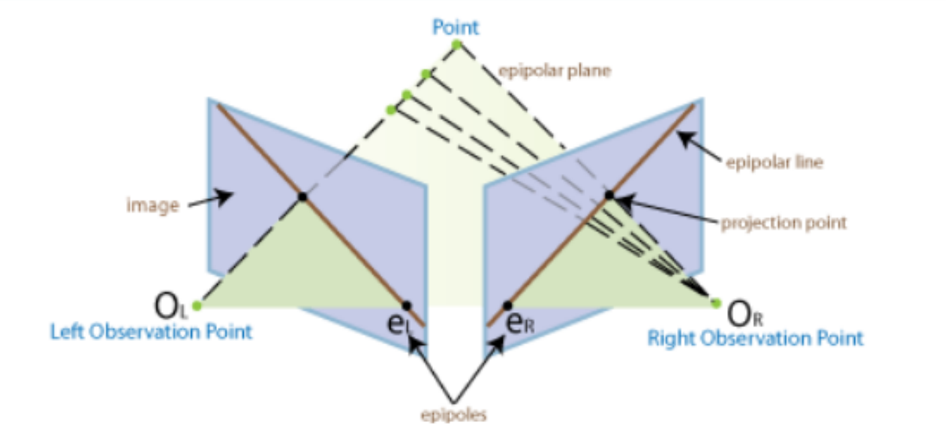
•The central controller can continuously or periodically poll various sensors on the CANbus network to collect real-time data.

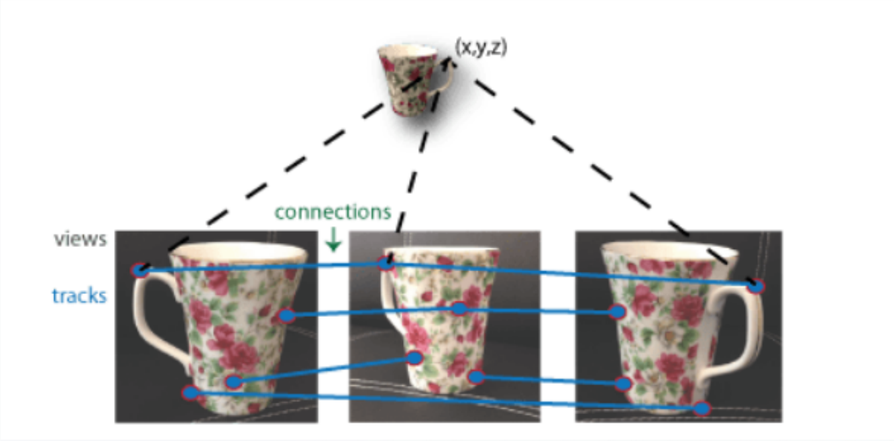
•Each sensor responds to the controller's requests with the current sensor data, which can include information such as temperature, pressure, speed, or any other parameter it is designed to measure.

•The controller can then use this data for various purposes, including system monitoring, control, or display.

SFM - STRUCTURE FROM MOTION

- SFM is a perception application that aims to reconstruct three-dimensional structures from a sequence of two-dimensional moving images. SfM is used in many applications, such as 3-D scanning, augmented reality, and mapping (vSLAM).
- SFM uses the subsequent images to triangulate the 3D position of objects in the environment.
- SfM requires point correspondences between images. Find corresponding points by matching features or tracking points from image 1 to image 2. The fundamental matrix relates the corresponding set of points in two images from different views.





LANE DETECTION

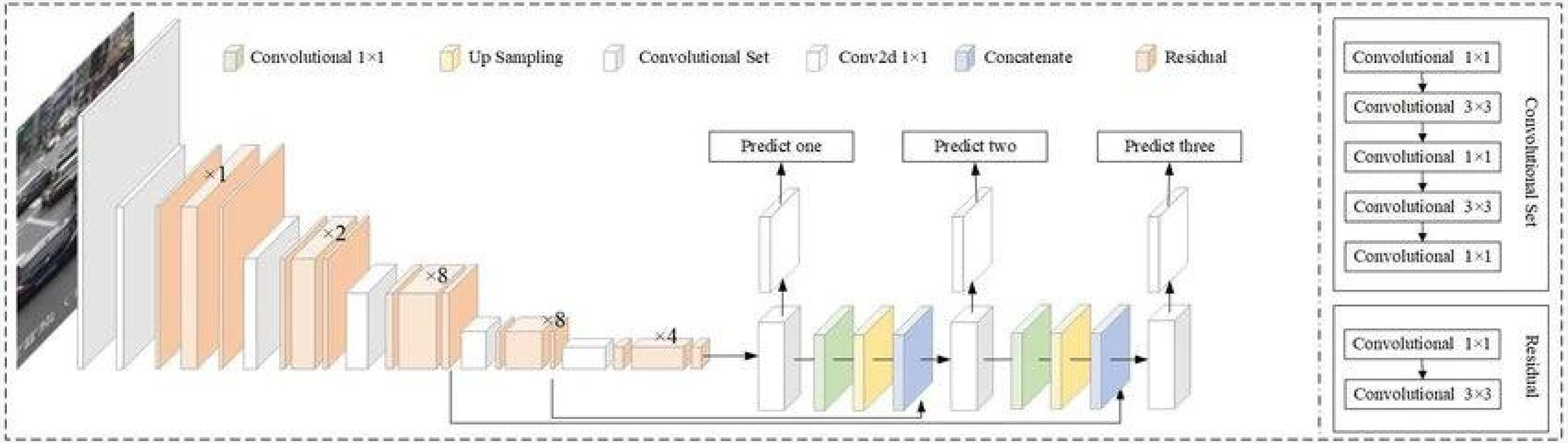
- Lane detection is a perception application that aims to detect road boundaries (lane line markings) and estimate the vehicle pose with respect to the detected lines using visual sensors on the vehicle.
- The application includes the localization of the road, the determination of the relative position between the vehicle and the road, and the analysis of the vehicle's heading direction.
- The steps followed for lane detection are:
  - Apply a distortion correction to raw images.
  - Use colour transforms to create a thresholded binary image.
  - Apply a perspective transform to generate a "bird's-eye view" of the image.
  - Detect lane pixels and fit to find the lane boundary.
  - Determine the curvature of the lane and vehicle position with respect to the centre.
  - Warp the detected lane boundaries back onto the original image.

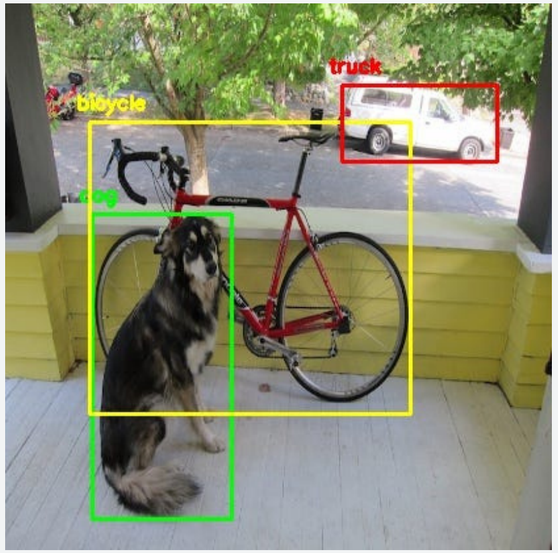
Applications in a Self-driving Vehicle

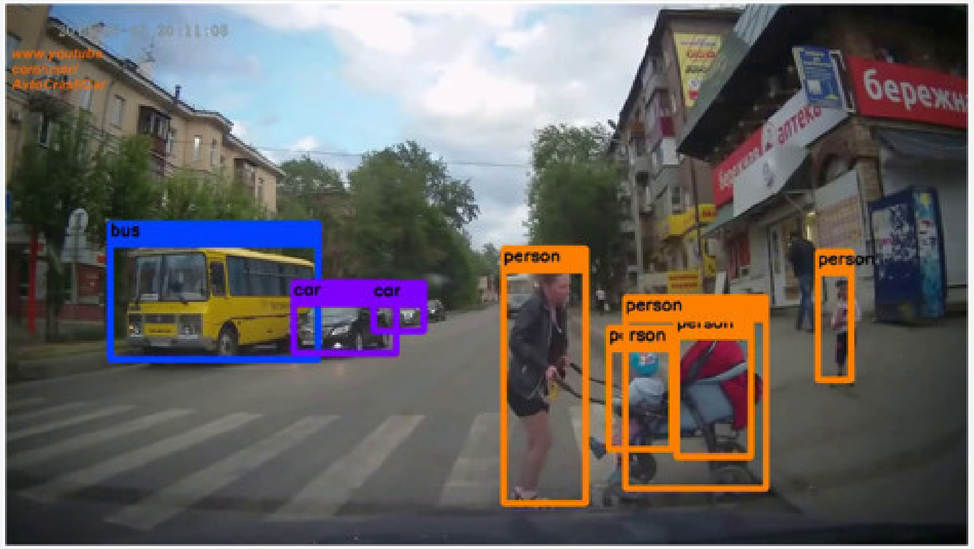
Application	Stage	Input	Output
Camera Grabber	➢ Sensing	➢ Packets over Automotive Ethernet	S1 ➢ Image frames in the shared memory
LIDAR/RADAR	➢ Sensing	➢ Packets over Automotive Ethernet	S2 ➢ Point cloud in the shared memory
CAN bus pooling	➢ Sensing	➢ Messages over CAN bus	S3 ➢ Sensed information in shared memory
Structure From Motion	➢ Perception	S1	P1 ➢ Depth Estimation
Lane Detection	➢ Perception	S1	P2 ➢ Lane Boundaries

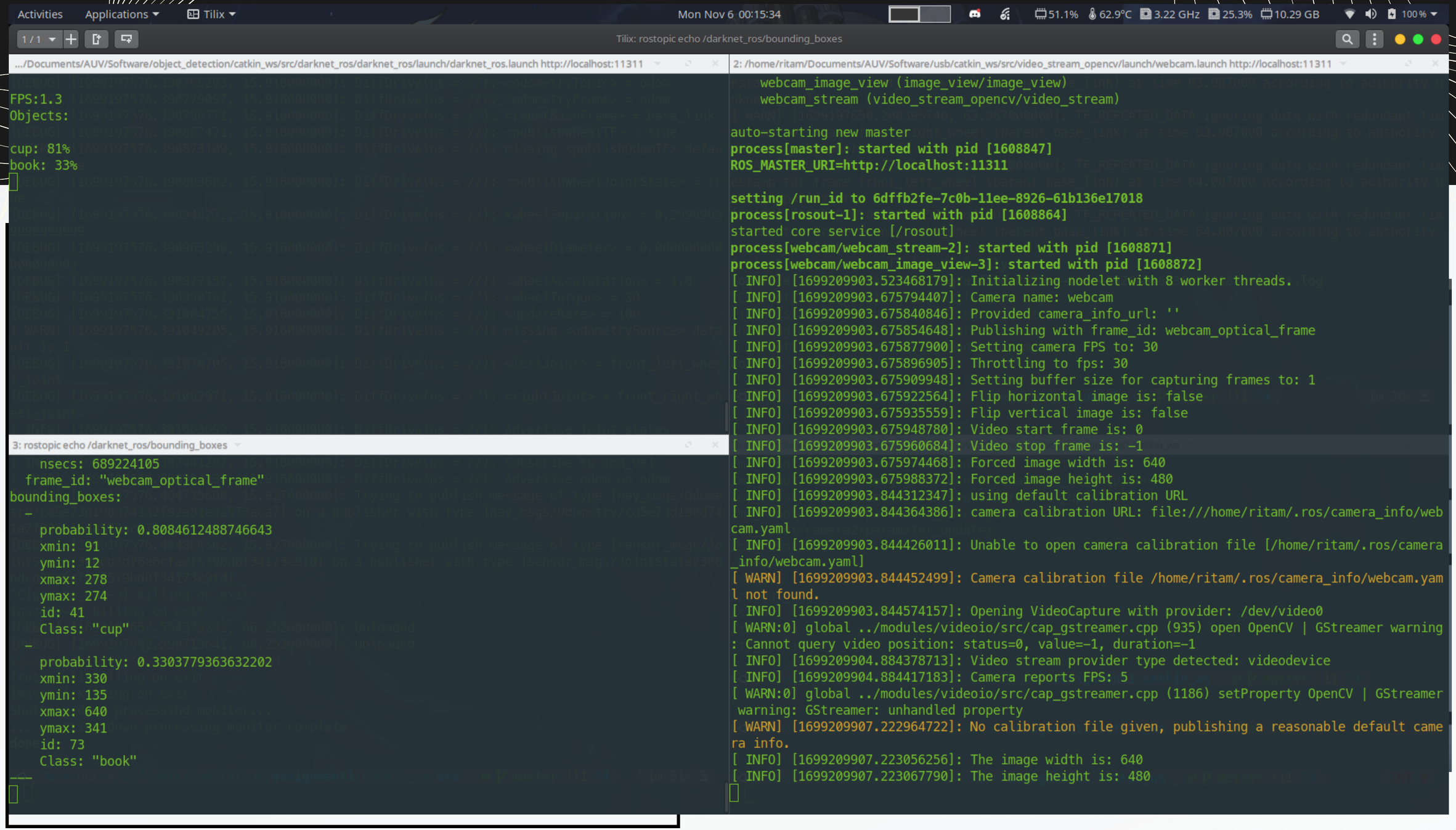
OBJECT DETECTION AND TRACKING

- Vision-based object detection is a perception application that is one of the primary prerequisites for self-driving vehicles. Distance and ranging sensors (e.g., LIDAR, RADAR) alone are insufficient to meet the requirements of self-driving.
- Modern object detection applications use neural networks along with visual sensor data to identify objects in the area surrounding the vehicle by drawing bounding boxes and classifying the objects inside each bounding box.
- Given some objects of interest marked in a frame, the object tracking application locates the objects in subsequent frames in the video.
- Object tracking tracks objects as they move in the environment. It also allows the self-driving vehicle to estimate the motion of objects and predict how they will move in the subsequent frames.
- YOLOv3 is a real-time object detection algorithm that identifies specific objects in videos, live feeds, or images. The YOLO machine learning algorithm uses features learned by a deep convolutional neural network to detect an object.


Convolutional 1×1
Up Sampling
Convolutional Set
Conv2d 1×1
Concatenate
Residual
×1
×2
×8
×8
×4
Predict one
Predict two
Predict three
Convolutional 1×1
Convolutional 3×3
Convolutional 1×1
Convolutional 3×3
Convolutional 1×1
Convolutional Set
Convolutional 1×1
Convolutional 3×3
Residual

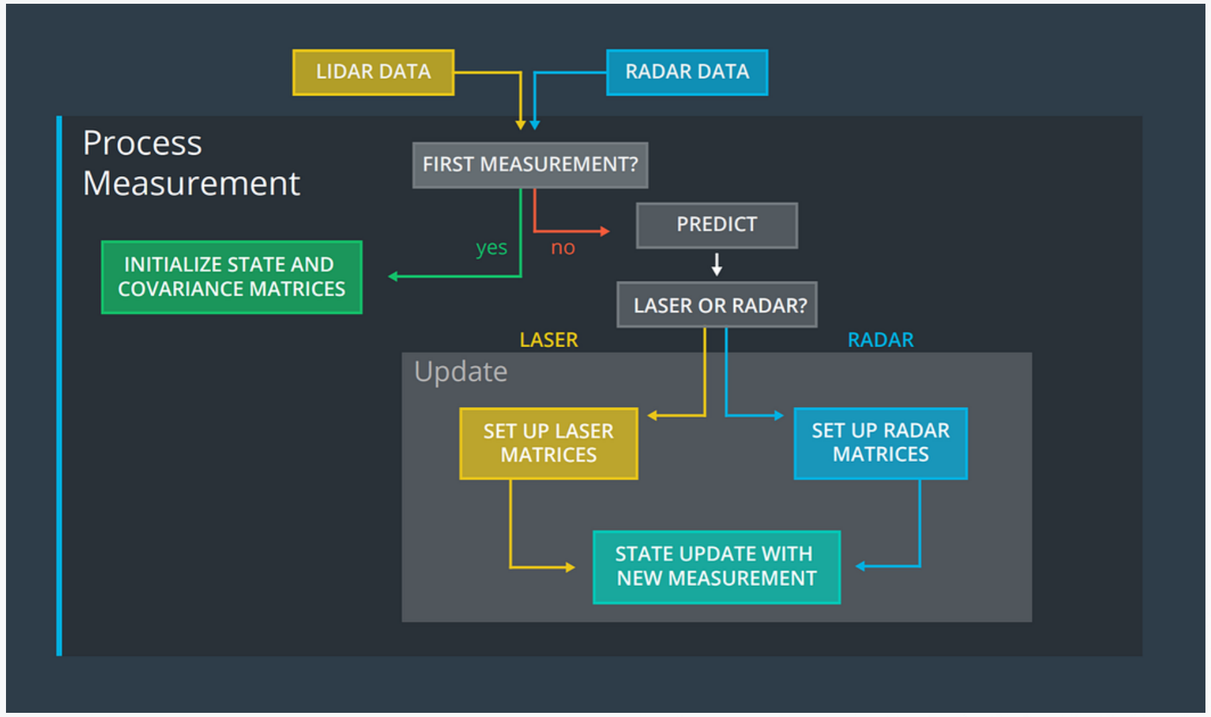

truck
bicycle
dog


bus
car
car
person
person
person
person


Activities
Applications
Tilix
Mon Nov 6 00:15:34
51.1%
62.9°C
3.22 GHz
25.3%
10.29 GB
100 %
1 / 1
Tilix: rostopic echo /darknet_ros/bounding_boxes
.../Documents/AUV/Software/object_detection/catkin_ws/src/darknet_ros/darknet_ros/launch/darknet_ros.launch http://localhost:11311
FPS:1.3
Objects:
cup: 81%
book: 33%
2: /home/ritam/Documents/AUV/Software/usb/catkin_ws/src/video_stream_opencv/launch/webcam.launch http://localhost:11311
webcam_image_view (image_view/image_view)
webcam_stream (video_stream_opencv/video_stream)
auto-starting new master
process[master]: started with pid [1608847]
ROS_MASTER_URI=http://localhost:11311
setting /run_id to 6dffb2fe-7c0b-11ee-8926-61b136e17018
process[rosout-1]: started with pid [1608864]
started core service [/rosout]
process[webcam/webcam_stream-2]: started with pid [1608871]
process[webcam/webcam_image_view-3]: started with pid [1608872]
[INFO] [1699209903.523468179]: Initializing nodelet with 8 worker threads.
[INFO] [1699209903.675794407]: Camera name: webcam
[INFO] [1699209903.675840846]: Provided camera_info_url: ''
[INFO] [1699209903.675854648]: Publishing with frame_id: webcam_optical_frame
[INFO] [1699209903.675877900]: Setting camera FPS to: 30
[INFO] [1699209903.675896905]: Throttling to fps: 30
[INFO] [1699209903.675909948]: Setting buffer size for capturing frames to: 1
[INFO] [1699209903.675922564]: Flip horizontal image is: false
[INFO] [1699209903.675935559]: Flip vertical image is: false
[INFO] [1699209903.675948780]: Video start frame is: 0
[INFO] [1699209903.675960684]: Video stop frame is: -1
[INFO] [1699209903.675974468]: Forced image width is: 640
[INFO] [1699209903.675988372]: Forced image height is: 480
[INFO] [1699209903.844312347]: using default calibration URL
[INFO] [1699209903.844364386]: camera calibration URL: file:///home/ritam/.ros/camera_info/webcam.yaml
[INFO] [1699209903.844426011]: Unable to open camera calibration file [/home/ritam/.ros/camera_info/webcam.yaml]
[WARN] [1699209903.844452499]: Camera calibration file /home/ritam/.ros/camera_info/webcam.yaml not found.
[INFO] [1699209903.844574157]: Opening VideoCapture with provider: /dev/video0
[WARN:0] global ../modules/videoio/src/cap_gstreamer.cpp (935) open OpenCV | GStreamer warning: Cannot query video position: status=0, value=-1, duration=-1
[INFO] [1699209904.884378713]: Video stream provider type detected: videodevice
[INFO] [1699209904.884417183]: Camera reports FPS: 5
[WARN:0] global ../modules/videoio/src/cap_gstreamer.cpp (1186) setProperty OpenCV | GStreamer warning: GStreamer: unhandled property
[WARN] [1699209907.222964722]: No calibration file given, publishing a reasonable default camera info.
[INFO] [1699209907.223056256]: The image width is: 640
[INFO] [1699209907.223067790]: The image height is: 480
3: rostopic echo /darknet_ros/bounding_boxes
nsecs: 689224105
frame_id: "webcam_optical_frame"
bounding_boxes:
-
probability: 0.8084612488746643
xmin: 91
ymin: 12
xmax: 278
ymax: 274
id: 41
Class: "cup"
-
probability: 0.3303779363632202
xmin: 330
ymin: 135
xmax: 640
ymax: 341
id: 73
Class: "book"

EXTENDED KALMAN FILTER (EKF)

- The result of localization (using distance sensors like RADAR) is prone to drift over time and can be noisy.
- A Kalman filter is an excellent candidate for combining distance information with other vehicle status information (e.g., IMU, Odometer, GPS) and handling such disturbances.
- Kalman filter fuses multiple data sources and performs continuous prediction (for missing data) and correction (for drifting data) on the localization results.
- The extended Kalman filter (EKF) is the nonlinear version of the Kalman filter.



FUSION

- The fusion task helps combine object information with the car's location on the fly. The information is sourced by pre-processing raw data from different sensors (e.g., cameras, different types of RADAR, LIDAR) and fused synchronously.

- The fusion process involves transforming different coordinate systems and updating the environment maps periodically in real-time.

PLANNING

Path Planner

- The planning stage is responsible for understanding higher-level goals from the user and converting them to purposeful decisions to achieve the higher-level goals while avoiding obstacles.

- The complexity of this stage compels a hierarchical design by partitioning the software into layers:
  - Mission planning: Mission planning computes the global trajectory based on the current location and the target destination
  - Behavioural planning: Behavioural planning generates local objectives
  - Motion planning: Motion planning takes the local objectives and generates the control plan to actuate the steering and speed.

DASM

- The **D**river **A**ssistance **S**ystem **M**odule performs the final actuation stage in the pipeline. The local objectives of the path planner are realized through a PID controller.

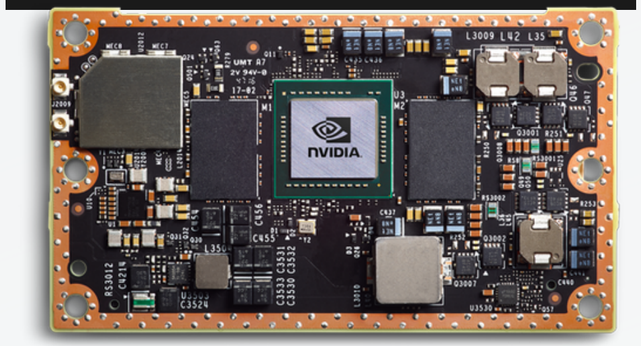
- The PID controller actuates the speed and steering based on the velocity and angle commands and can automatically use the odometer and IMU feedback to maintain the targets set by the path planning stage.

Applications in a Self-driving Vehicle

Application	Stage	Input	Output
Object Detection	➢ Perception	S1, S2	I1 ➢ Bounding Box
Object Tracking	➢ Perception	I1	I2 ➢ Object Movement
Localization	➢ Perception	S2	I3 ➢ Position and orientation pose
Extended Kalman Filter	➢ Perception	I3, S3	I4 ➢ Corrected pose
Fusion	➢ Perception	I2, I4	P3 ➢ Fused object and vehicle location
Path Planner	➢ Planning	P1, P2, P3	A1 ➢ Spatio-temporal trajectory
Driver Assistance System Module	➢ Actuation	A1	➢ Steer, Brake

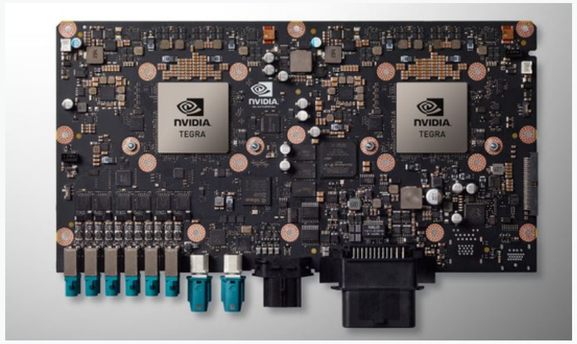
SUPPORTED PLATFORMS

JETSON TX2 GPU



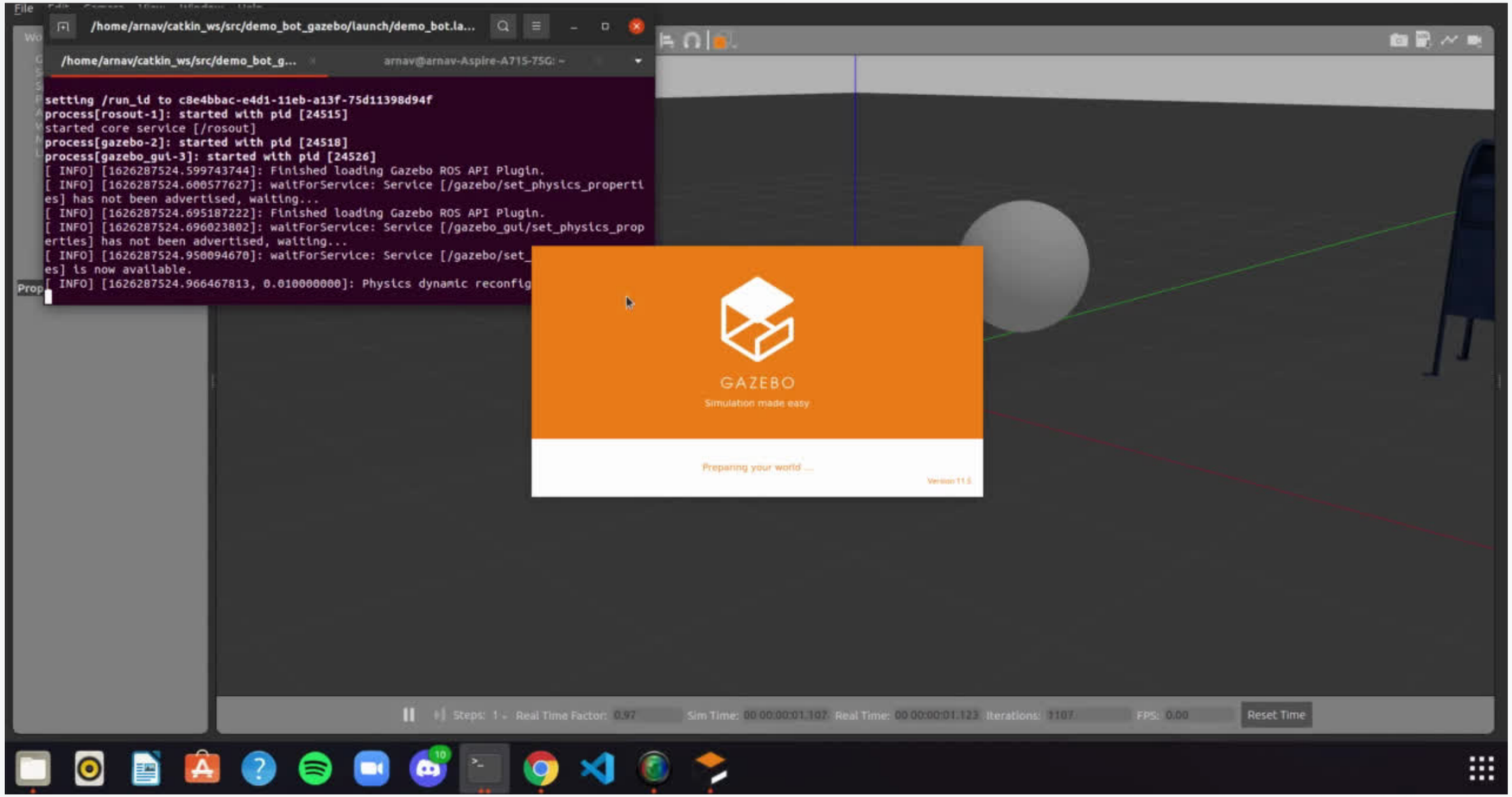
- 8 GB 128-bit LPDDR4 Memory
- 7.5 Watt
- 59.7 GB/s of memory bandwidth
- 1.33 TFLOPS

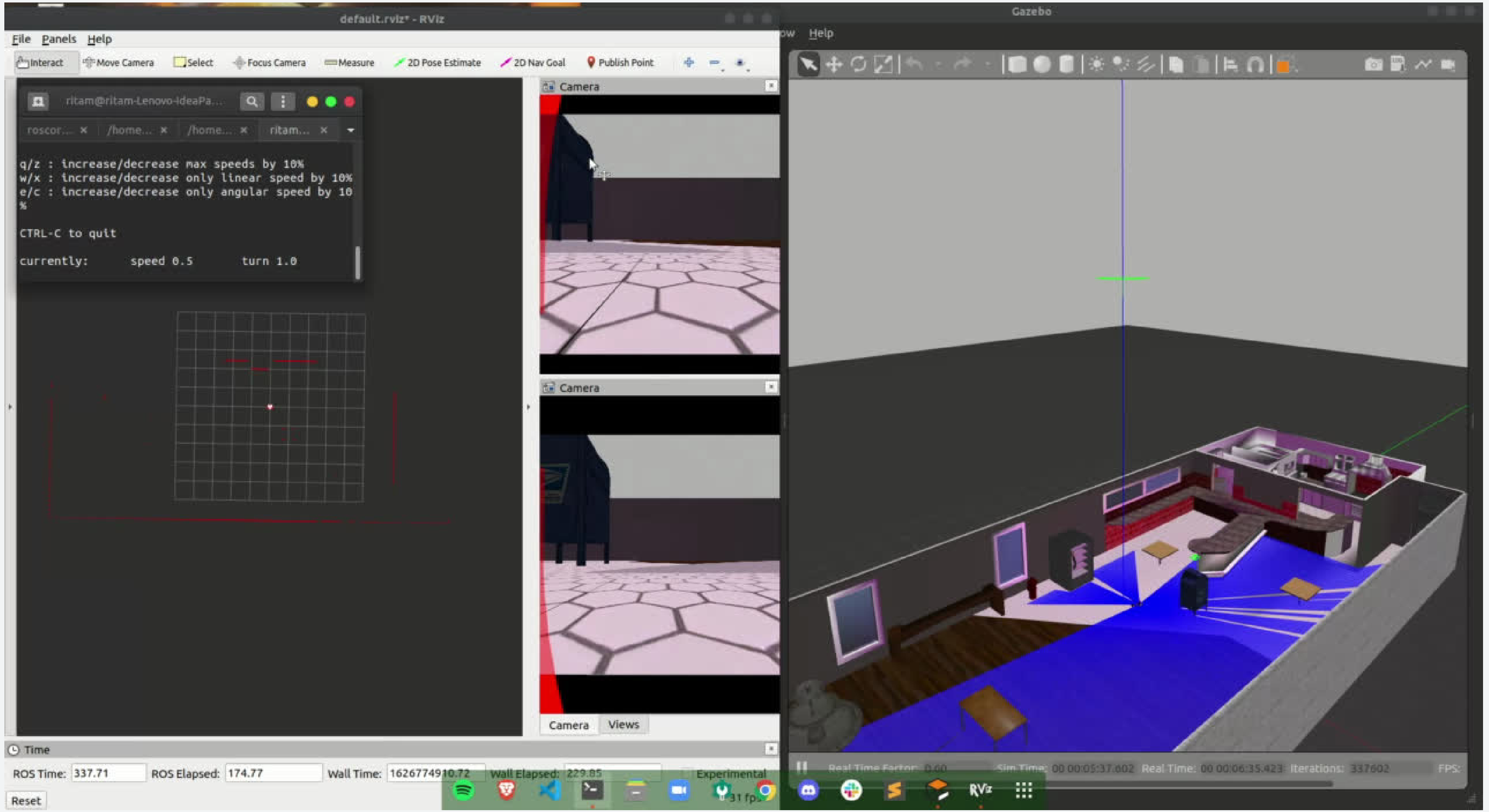
NVIDIA Drive PX2

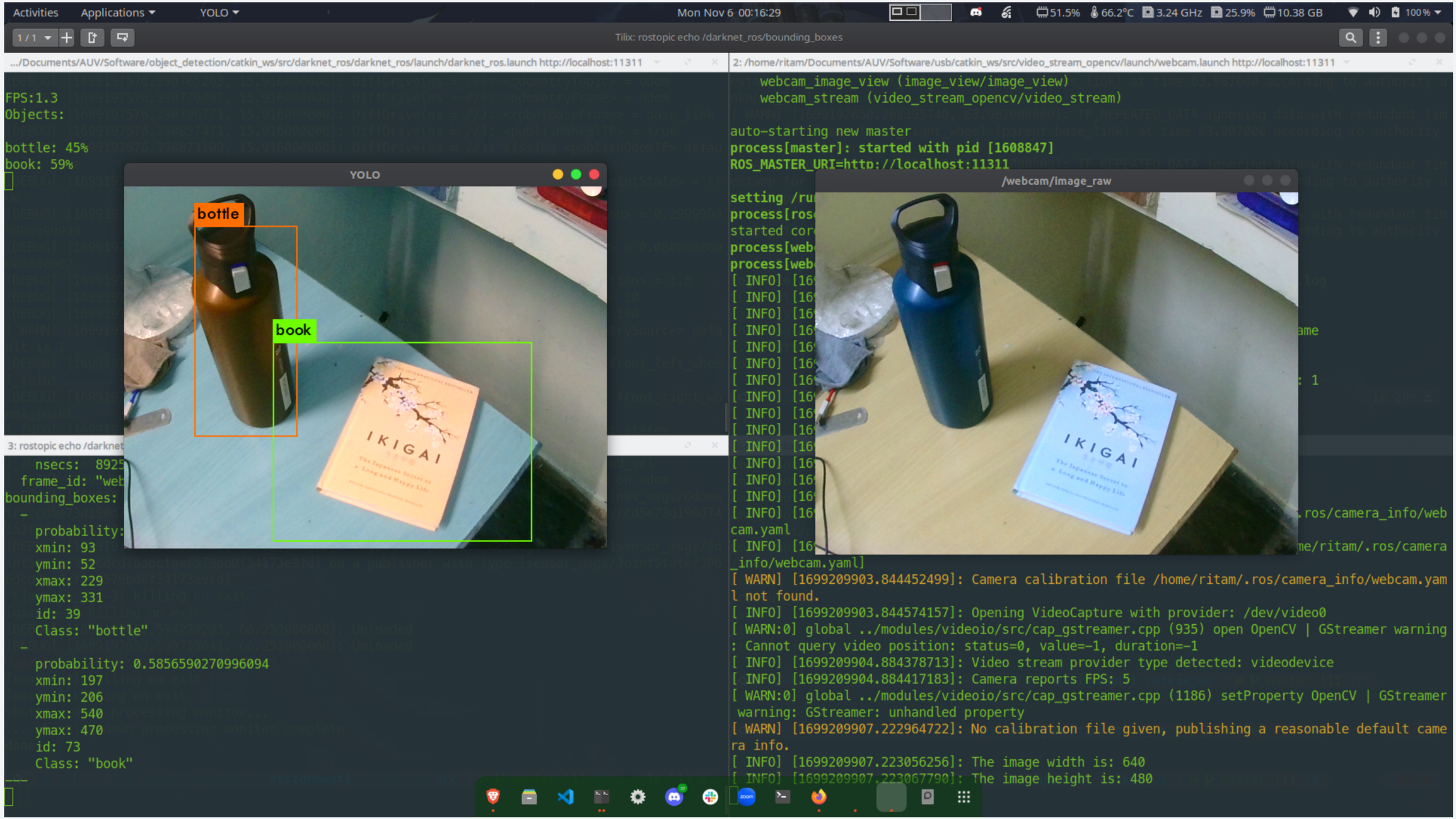


- No cross-compiler support
- 250 Watt Total Dissipated Power (TDP)
- 12 core ARM CPU (in total)

IMPLEMENTATION




default.rviz* - RViz
File Panels Help
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
ritam@ritam-Lenovo-IdeaPa...
roscor...
/home...
/home...
ritam...
q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10
%
CTRL-C to quit
currently: speed 0.5 turn 1.0
Camera
Camera
Camera
Views
Time
ROS Time: 337.71
ROS Elapsed: 174.77
Wall Time: 1626774910.72
Wall Elapsed: 229.85
Experimental
Reset
Gazebo
Help
Real Time Factor: 0.00
Sim Time: 00 00:05:37.602
Real Time: 00 00:06:35.423
Iterations: 337602
FPS:


Activities
Applications
YOLO
Mon Nov 6 00:16:29
51.5% 66.2°C 3.24 GHz 25.9% 10.38 GB
100 %
Tilix: rostopic echo /darknet_ros/bounding_boxes
.../Documents/AUV/Software/object_detection/catkin_ws/src/darknet_ros/darknet_ros/launch/darknet_ros.launch http://localhost:11311
FPS:1.3
Objects:
bottle: 45%
book: 59%
YOLO
bottle
book
IKIGAI
2: /home/ritam/Documents/AUV/Software/usb/catkin_ws/src/video_stream_opencv/launch/webcam.launch http://localhost:11311
webcam_image_view (image_view/image_view)
webcam_stream (video_stream_opencv/video_stream)
auto-starting new master
process[master]: started with pid [1608847]
ROS_MASTER_URI=http://localhost:11311
/webcam/image_raw
IKIGAI
3: rostopic echo /darknet
nsecs: 8925
frame_id: "web
bounding_boxes:
probability:
xmin: 93
ymin: 52
xmax: 229
ymax: 331
id: 39
Class: "bottle"
probability: 0.5856590270996094
xmin: 197
ymin: 206
xmax: 540
ymax: 470
id: 73
Class: "book"
[INFO] [1699209903.844574157]: Opening VideoCapture with provider: /dev/video0
[WARN:0] global ../modules/videoio/src/cap_gstreamer.cpp (935) open OpenCV | GStreamer warning: Cannot query video position: status=0, value=-1, duration=-1
[INFO] [1699209904.884378713]: Video stream provider type detected: videodevice
[INFO] [1699209904.884417183]: Camera reports FPS: 5
[WARN:0] global ../modules/videoio/src/cap_gstreamer.cpp (1186) setProperty OpenCV | GStreamer warning: GStreamer: unhandled property
[WARN] [1699209907.222964722]: No calibration file given, publishing a reasonable default camera info.
[INFO] [1699209907.223056256]: The image width is: 640
[INFO] [1699209907.223067790]: The image height is: 480

Definition of the SLAM Problem

Given

- The robot's controls

$$u_{1:T} = \{u_1, u_2, u_3, \ldots, u_T\}$$

- Observations

$$z_{1:T} = \{z_1, z_2, z_3, \ldots, z_T\}$$

Wanted

- Map of the environment

$$m$$

- Path of the robot

$$x_{0:T} = \{x_0, x_1, x_2, \ldots, x_T\}$$

Ref. : http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/pdf/slam04-ekf.pdf

Kalman Filter Algorithm

1: **Kalman_filter**$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$**:**

2: $\bar{\mu}_t = A_t\ \mu_{t-1} + B_t\ u_t$

3: $\bar{\Sigma}_t = A_t\ \Sigma_{t-1}\ A_t^T + R_t$

4: $K_t = \bar{\Sigma}_t\ C_t^T (C_t\ \bar{\Sigma}_t\ C_t^T + Q_t)^{-1}$

5: $\mu_t = \bar{\mu}_t + K_t(z_t - C_t\ \bar{\mu}_t)$

6: $\Sigma_t = (I - K_t\ C_t)\ \bar{\Sigma}_t$

7: *return* μ_t, Σ_t

Ref. : http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/pdf/slam04-ekf.pdf

SUMMARY

We implemented a simulation of a bot in a gezebo world with cameras, LIDAR to map and track the environment.

We implemented a YOLOv3 based object detection and tracking which was able to identify objects with an accuracy of 91%

We also implemented a Kalman filter-based SLAM algorithm to map a path avoiding obstacles from source to destination.

THANK YOU