Multi-Object Tracking Project

Outline

- Introduction
- Project Overview & Objectives
- Design Process & Challenges
- Implementation Details & Challenges
- Demo
- Conclusion

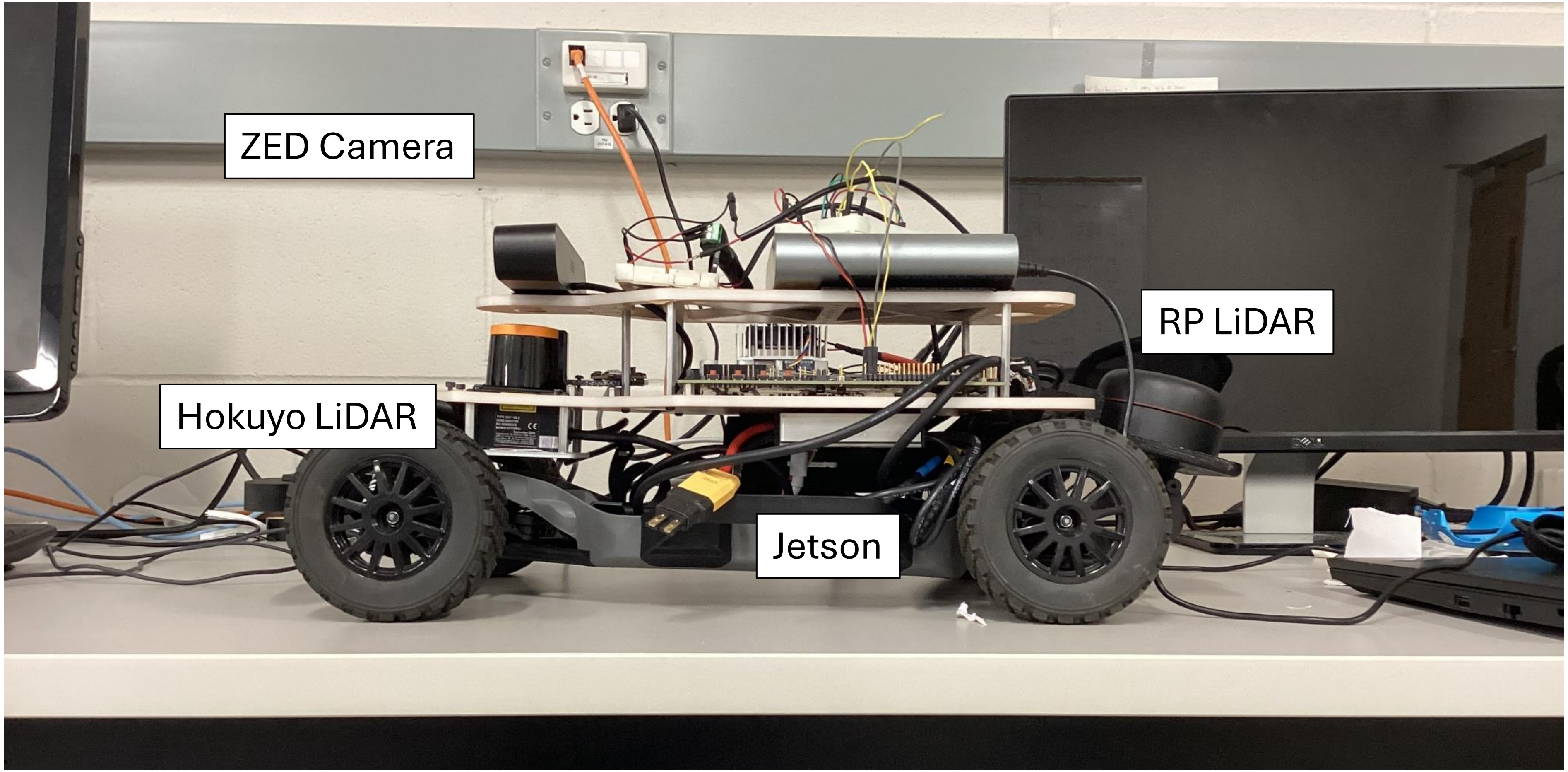
Introduction

Introduction

Introduction

https://mcscert.mcmaster.ca/

Overview



Overview

Could we perform object **tracking** and **classification** with a 2D LiDAR and an RGB camera?

Overview

Could we perform object **tracking** and **classification** with a 2D LiDAR and an RGB camera?

→ ??? →

Track 1
Track 2
...
Track N

Track {
id
position
velocity
class
...
}

Objective

To **design** and **implement** a multi-object tracking and classification pipeline in the RC car

Objective

~~To **design** and **implement** a multi-object tracking and classification pipeline in the RC car~~

To **learn as much as I can** about perception, and demonstrate my learning

Objective

~~To **design** and **implement** a multi-object tracking and classification pipeline in the RC car~~

To **learn as much as I can** about perception, and demonstrate my learning

4-months

Objective

~~To **design** and **implement** a multi-object tracking and classification pipeline in the RC car~~

To **learn as much as I can** about perception, and demonstrate my learning

part-time

4-months

Objective

~~To **design** and **implement** a multi-object tracking and classification pipeline in the RC car~~

To **learn as much as I can** about perception, and demonstrate my learning

part-time

4-months

little robotics experience

Objective

~~To **design** and **implement** a multi-object tracking and classification pipeline in the RC car~~

To **learn as much as I can** about perception, and demonstrate my learning

part-time

~~efficient implementation~~

get SOMETHING working

4-months

little robotics experience

Part 1: Design

Sensor Input

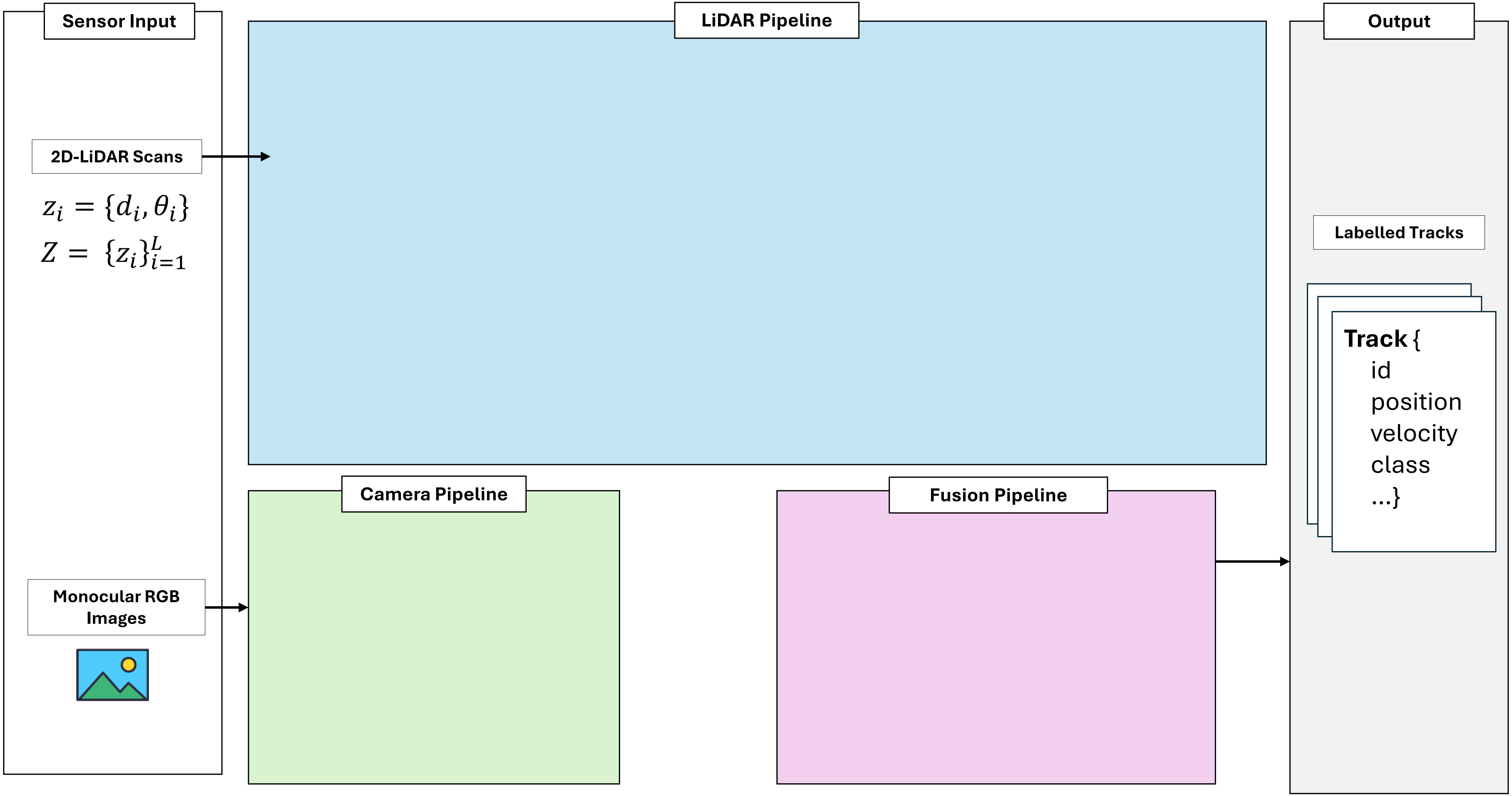
2D-LiDAR Scans

$$z_i = \{d_i, \theta_i\}$$

$$Z = \{z_i\}_{i=1}^{L}$$

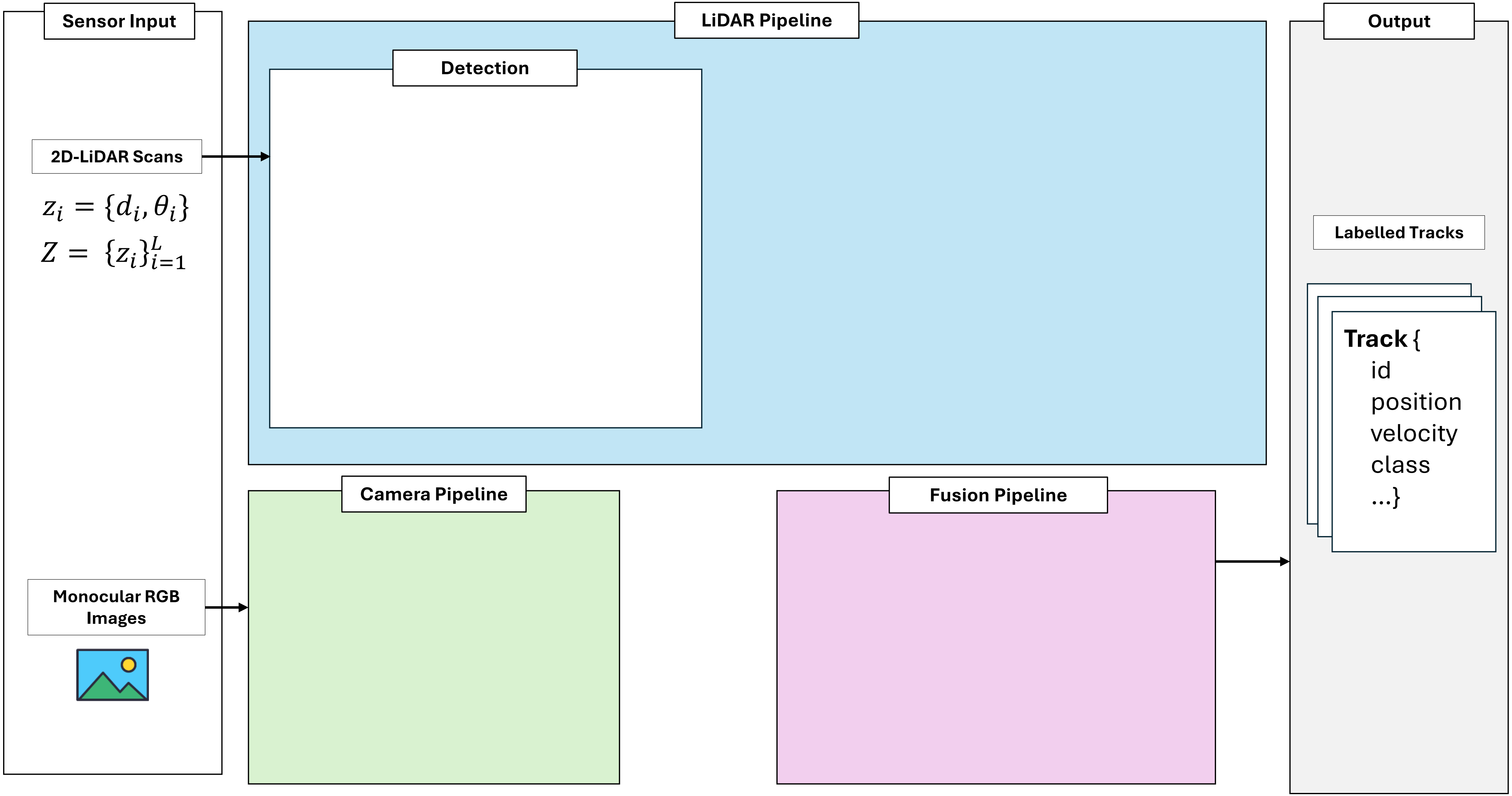
Monocular RGB Images

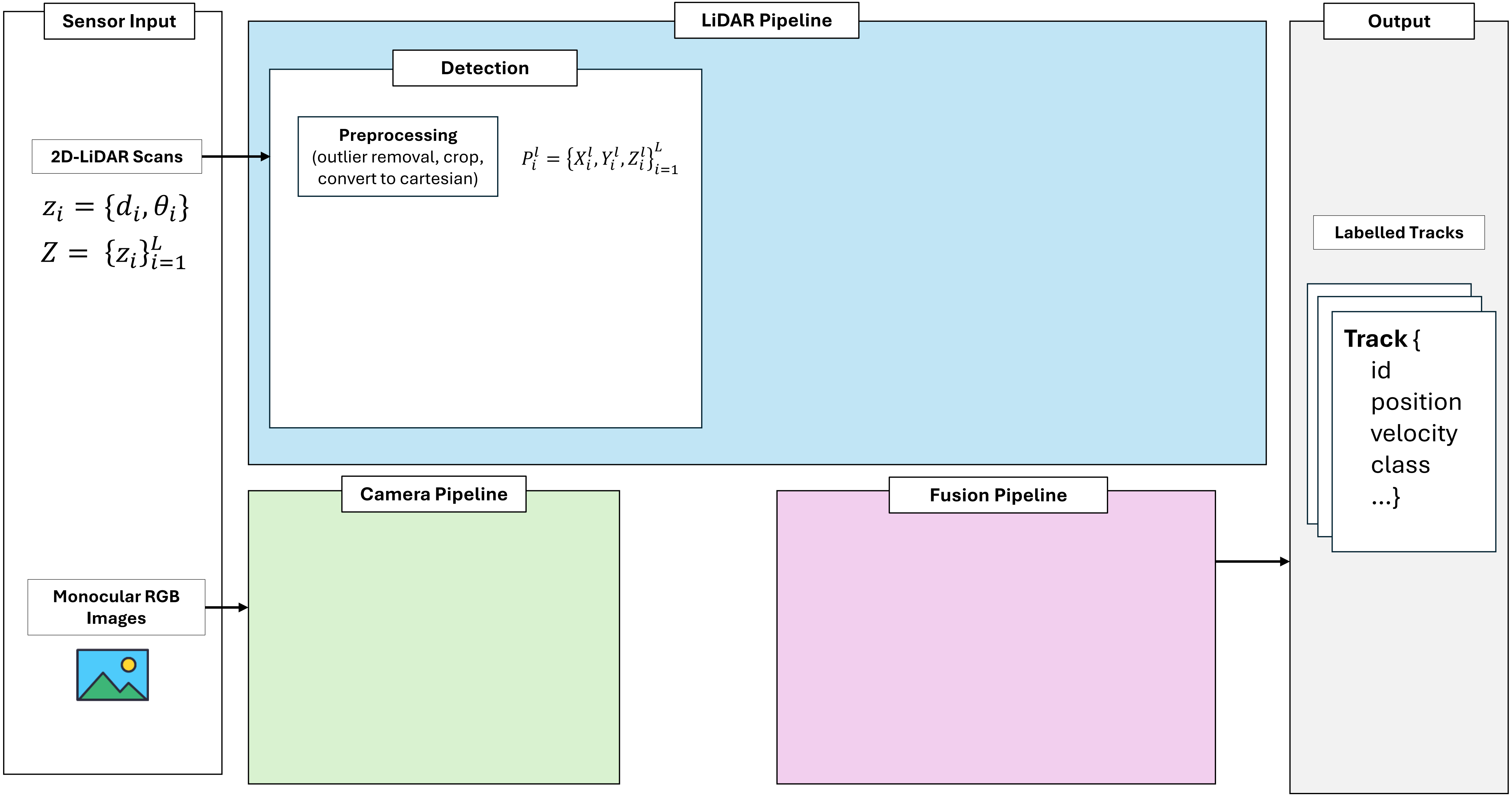

Sensor Input
2D-LiDAR Scans
$z_i = \{d_i, \theta_i\}$
$Z = \{z_i\}_{i=1}^{L}$
Monocular RGB Images
Output
Labelled Tracks
Track {
id
position
velocity
class
...}

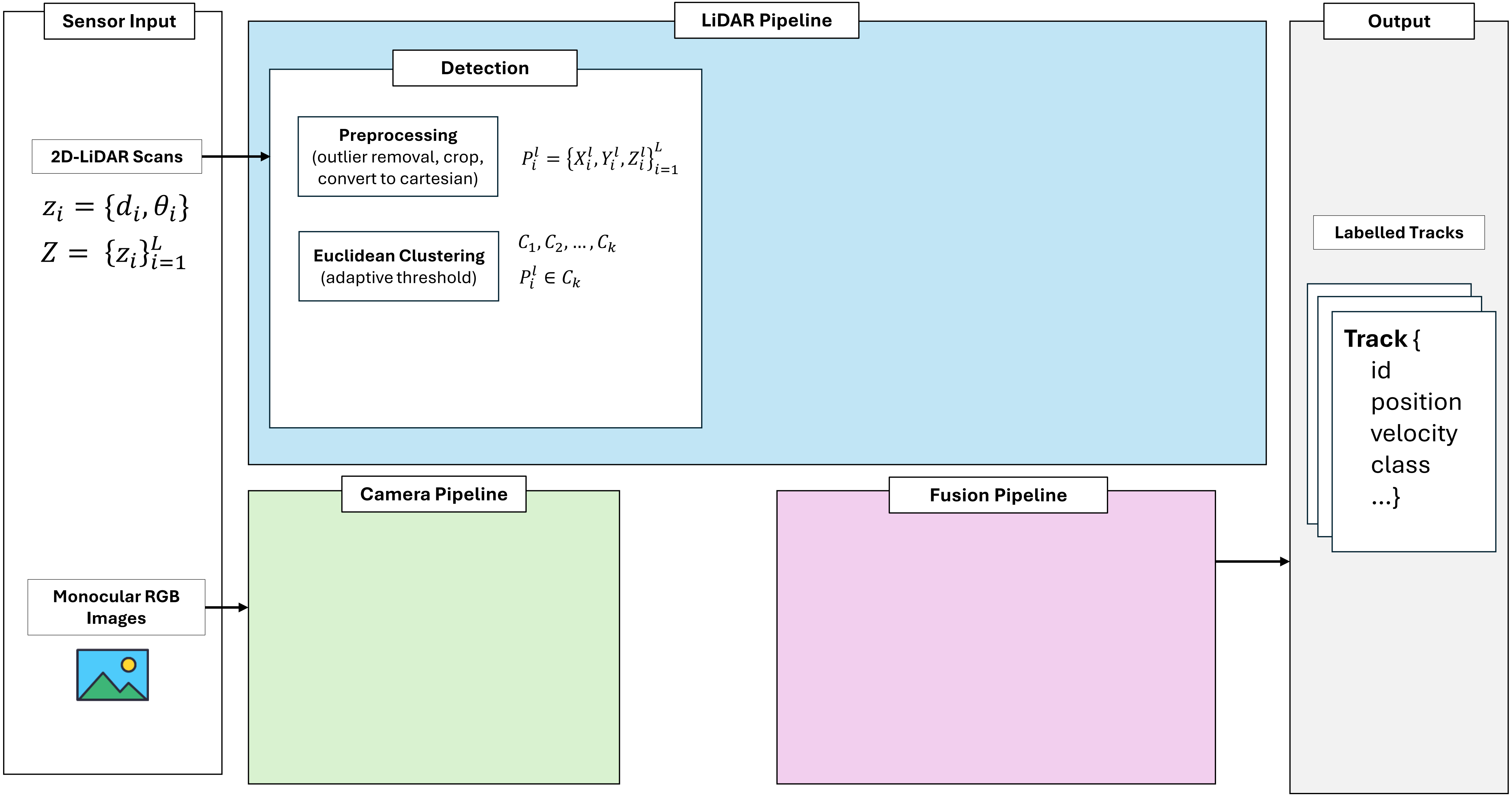

Sensor Input
LiDAR Pipeline
Output
2D-LiDAR Scans
$z_i = \{d_i, \theta_i\}$
$Z = \{z_i\}_{i=1}^{L}$
Labelled Tracks
Track {
id
position
velocity
class
...}
Camera Pipeline
Fusion Pipeline
Monocular RGB Images

Design Challenge 1: Knowledge gap

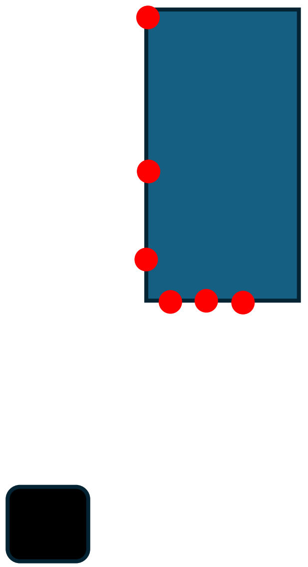
What is the pinhole camera model?
What does it mean to calibrate a camera?
Homogenous coordinates?
Transformation matrices?
...
+ more stuff I had to learn as I was designing


Sensor Input
2D-LiDAR Scans
$z_i = \{d_i, \theta_i\}$
$Z = \{z_i\}_{i=1}^{L}$
Monocular RGB Images
LiDAR Pipeline
Detection
Camera Pipeline
Fusion Pipeline
Output
Labelled Tracks
Track {
id
position
velocity
class
...}


Sensor Input
2D-LiDAR Scans
$z_i = \{d_i, \theta_i\}$
$Z = \{z_i\}_{i=1}^{L}$
Monocular RGB Images
LiDAR Pipeline
Detection
Preprocessing
(outlier removal, crop, convert to cartesian)
$P_i^l = \{X_i^l, Y_i^l, Z_i^l\}_{i=1}^{L}$
Camera Pipeline
Fusion Pipeline
Output
Labelled Tracks
Track {
id
position
velocity
class
...}


Sensor Input
2D-LiDAR Scans
$z_i = \{d_i, \theta_i\}$
$Z = \{z_i\}_{i=1}^{L}$
Monocular RGB Images
LiDAR Pipeline
Detection
Preprocessing (outlier removal, crop, convert to cartesian)
$P_i^l = \{X_i^l, Y_i^l, Z_i^l\}_{i=1}^{L}$
Euclidean Clustering (adaptive threshold)
$C_1, C_2, \ldots, C_k$
$P_i^l \in C_k$
Camera Pipeline
Fusion Pipeline
Output
Labelled Tracks
Track {
id
position
velocity
class
...}

Design Challenge 2: Sparse LiDAR points further away



Sensor Input

LiDAR Pipeline

Output

2D-LiDAR Sca

Monocular RG
Images

elled Tracks

ack {
id
position
velocity
class
...}

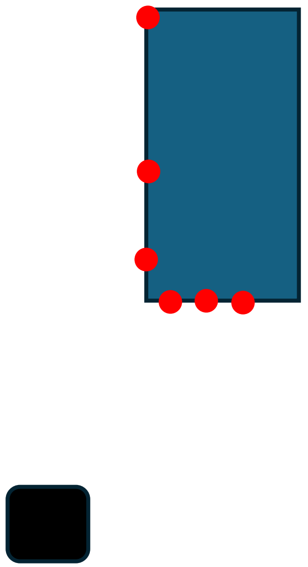
Sensor Input

LiDAR Pipeline

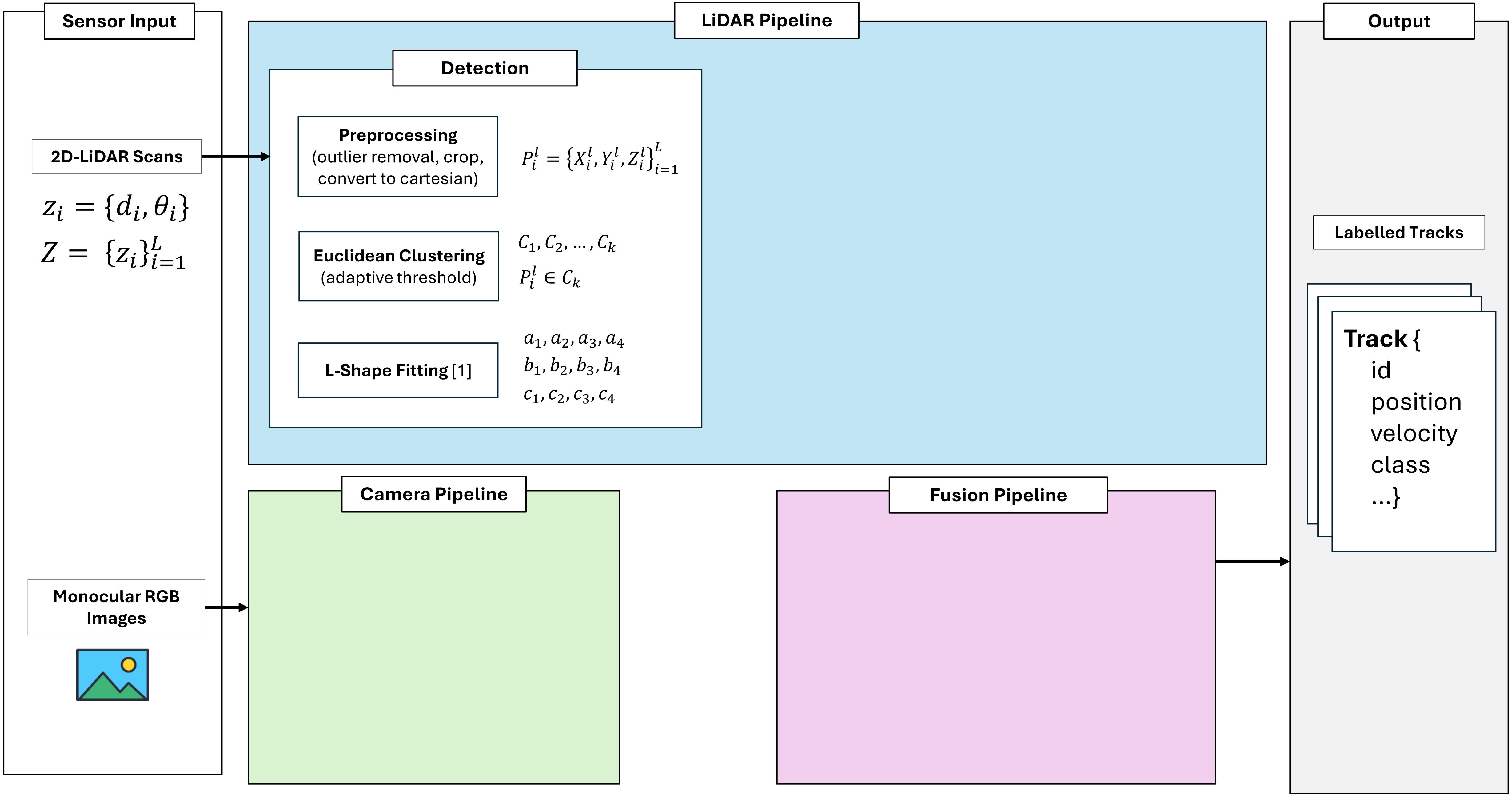
Output

Design Challenge 2: Sparse LiDAR points further away

$$r_i = r_0 + d_i * constant$$



Solution: use a threshold that increases as points are further away



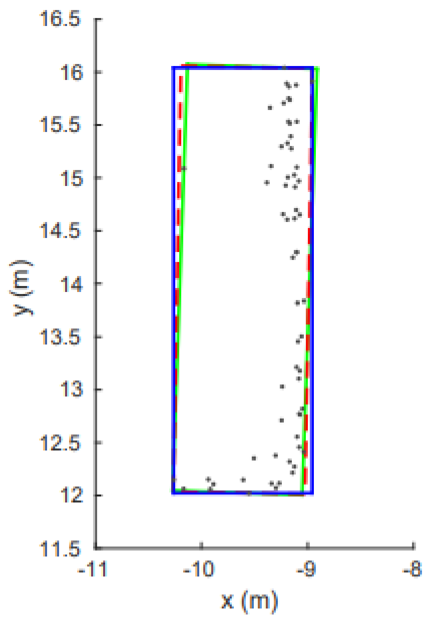
Sensor Input
2D-LiDAR Scans
$z_i = \{d_i, \theta_i\}$
$Z = \{z_i\}_{i=1}^{L}$
Monocular RGB Images
LiDAR Pipeline
Detection
Preprocessing (outlier removal, crop, convert to cartesian)
$P_i^l = \{X_i^l, Y_i^l, Z_i^l\}_{i=1}^{L}$
Euclidean Clustering (adaptive threshold)
$C_1, C_2, \dots, C_k$
$P_i^l \in C_k$
L-Shape Fitting [1]
a_1, a_2, a_3, a_4
b_1, b_2, b_3, b_4
c_1, c_2, c_3, c_4
Camera Pipeline
Fusion Pipeline
Output
Labelled Tracks
Track {
id
position
velocity
class
...}

L-Shape Fitting [1]

Algorithm 2 Search-Based Rectangle Fitting

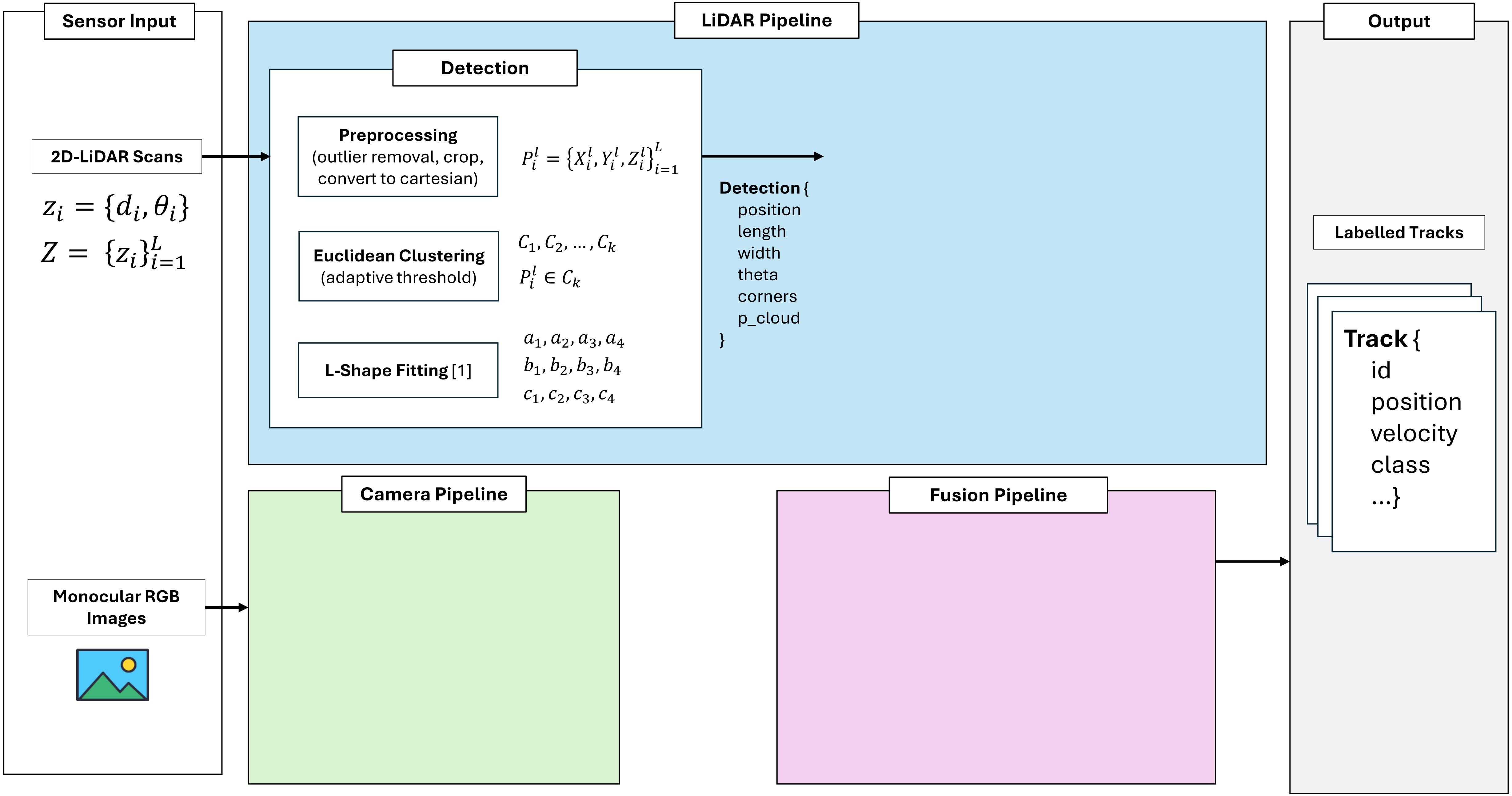
Input: range data points $X \in R^{n \times 2}$
Output: rectangle edges $\{a_i x + b_i x = c_i | i = 1, 2, 3, 4\}$

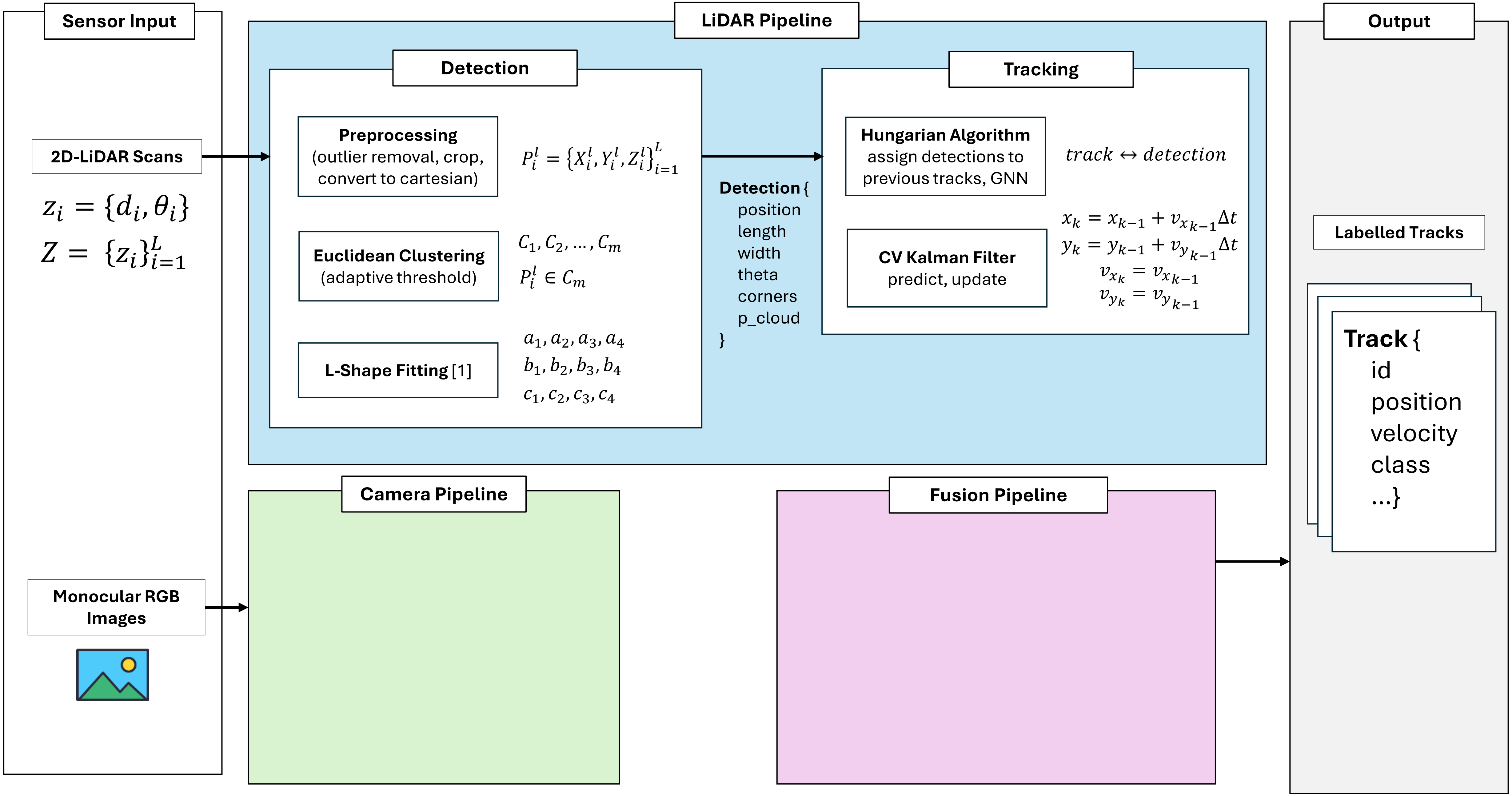
```
Q ← ∅
for θ = 0 to π/2 − δ step δ do
    ê1 ← (cos θ, sin θ)                        ▷ rectangle edge direction vector
    ê2 ← (− sin θ, cos θ)
    C1 ← X · ê1^T                              ▷ projection on to the edge
    C2 ← X · ê2^T
    q ← CalculatecriterionX(C1, C2)
    insert q into Q with key (θ)
end for
select key (θ*) from Q with maximum value
C1* ← X · (cos θ*, sin θ*)^T, C2* ← X · (− sin θ*, cos θ*)^T
a1 ← cos θ*,   b1 ← sin θ*,  c1 ← min{C1*}
a2 ← − sin θ*, b2 ← cos θ*,  c2 ← min{C2*}
a3 ← cos θ*,   b3 ← sin θ*,  c3 ← max{C1*}
a4 ← − sin θ*, b4 ← cos θ*,  c4 ← max{C2*}
```



Search based algorithm to fit a rectangle to a cluster of points

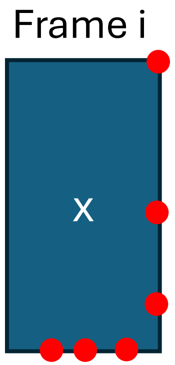
[1] X. Zhang, W. Xu, C. Dong and J. M. Dolan, "Efficient L-shape fitting for vehicle detection using laser scanners," 2017 IEEE Intelligent Vehicles Symposium (IV), Los Angeles, CA, USA, 2017, pp. 54-59, doi: 10.1109/IVS.2017.7995698. keywords: {Feature extraction;Shape;Laser radar;Three-dimensional displays;Clustering algorithms;Vehicle detection;Automobiles},


Sensor Input
2D-LiDAR Scans
$z_i = \{d_i, \theta_i\}$
$Z = \{z_i\}_{i=1}^{L}$
Monocular RGB Images
LiDAR Pipeline
Detection
Preprocessing (outlier removal, crop, convert to cartesian)
$P_i^l = \{X_i^l, Y_i^l, Z_i^l\}_{i=1}^{L}$
Euclidean Clustering (adaptive threshold)
$C_1, C_2, \ldots, C_k$
$P_i^l \in C_k$
L-Shape Fitting [1]
a_1, a_2, a_3, a_4
b_1, b_2, b_3, b_4
c_1, c_2, c_3, c_4
Detection { position length width theta corners p_cloud }
Camera Pipeline
Fusion Pipeline
Output
Labelled Tracks
Track { id position velocity class ...}

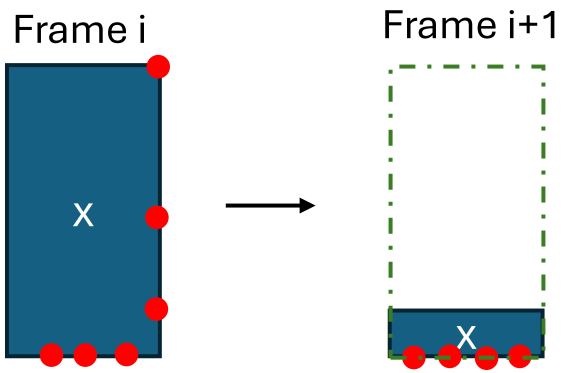


Sensor Input
2D-LiDAR Scans
$z_i = \{d_i, \theta_i\}$
$Z = \{z_i\}_{i=1}^{L}$
Monocular RGB Images
LiDAR Pipeline
Detection
Preprocessing (outlier removal, crop, convert to cartesian)
$P_i^l = \{X_i^l, Y_i^l, Z_i^l\}_{i=1}^{L}$
Euclidean Clustering (adaptive threshold)
$C_1, C_2, \ldots, C_m$
$P_i^l \in C_m$
L-Shape Fitting [1]
a_1, a_2, a_3, a_4
b_1, b_2, b_3, b_4
c_1, c_2, c_3, c_4
Detection { position length width theta corners p_cloud }
Tracking
Hungarian Algorithm assign detections to previous tracks, GNN
$track \leftrightarrow detection$
CV Kalman Filter predict, update
$x_k = x_{k-1} + v_{x_{k-1}} \Delta t$
$y_k = y_{k-1} + v_{y_{k-1}} \Delta t$
$v_{x_k} = v_{x_{k-1}}$
$v_{y_k} = v_{y_{k-1}}$
Camera Pipeline
Fusion Pipeline
Output
Labelled Tracks
Track { id position velocity class ...}

Design Challenge 3: Large jumps in rectangle



Design Challenge 3: Large jumps in rectangle

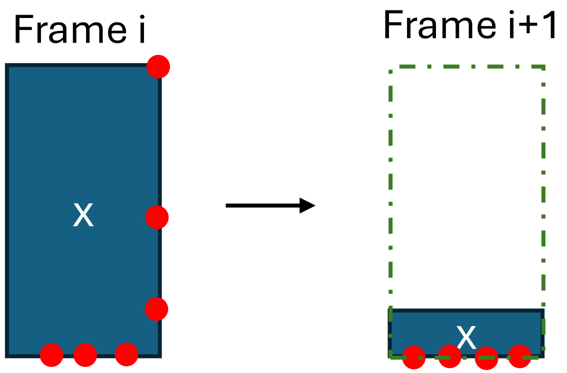


Sensor Input

LiDAR Pipeline

Output

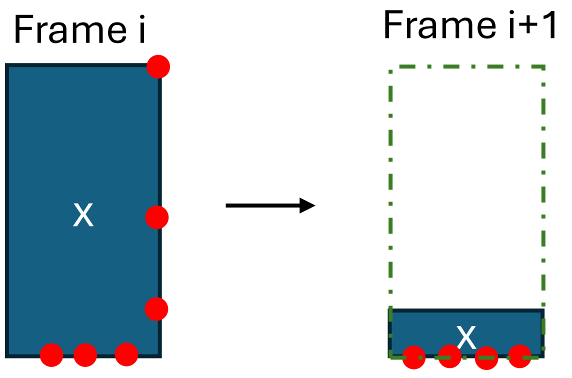
Design Challenge 3: Large jumps in rectangle



Solution 1: Use the largest length & width between the rectangles, recalculate center

e.g. curr_frame.length() = max(curr_frame.length(), prev_frame.length())

Design Challenge 3: Large jumps in rectangle

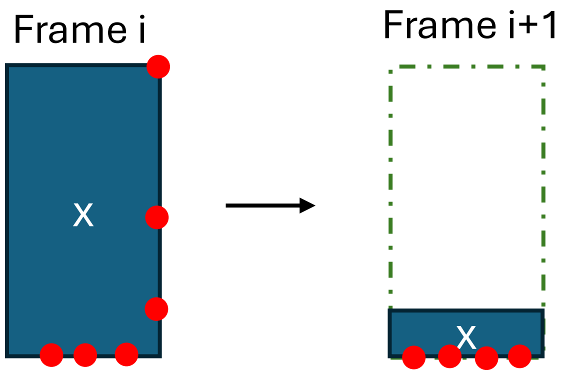


Solution 1: Use the largest length & width between the rectangles, recalculate center

e.g. curr_frame.length() = max(curr_frame.length(), prev_frame.length())

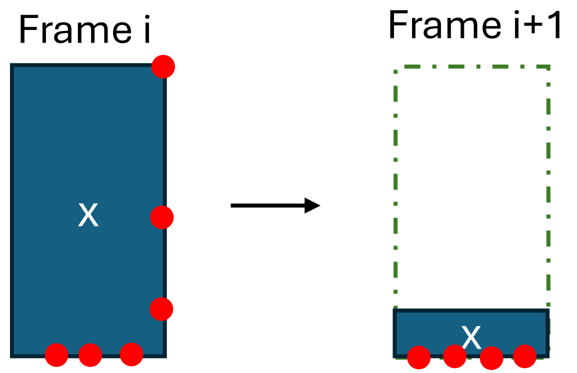
But what if two objects get fused together??

Design Challenge 3: Large jumps in rectangle



Solution 2: Keep a circular queue of last n associated detections, take the median

Design Challenge 3: Large jumps in rectangle



Solution 3: Use a Kalman Filter to estimate the length and width

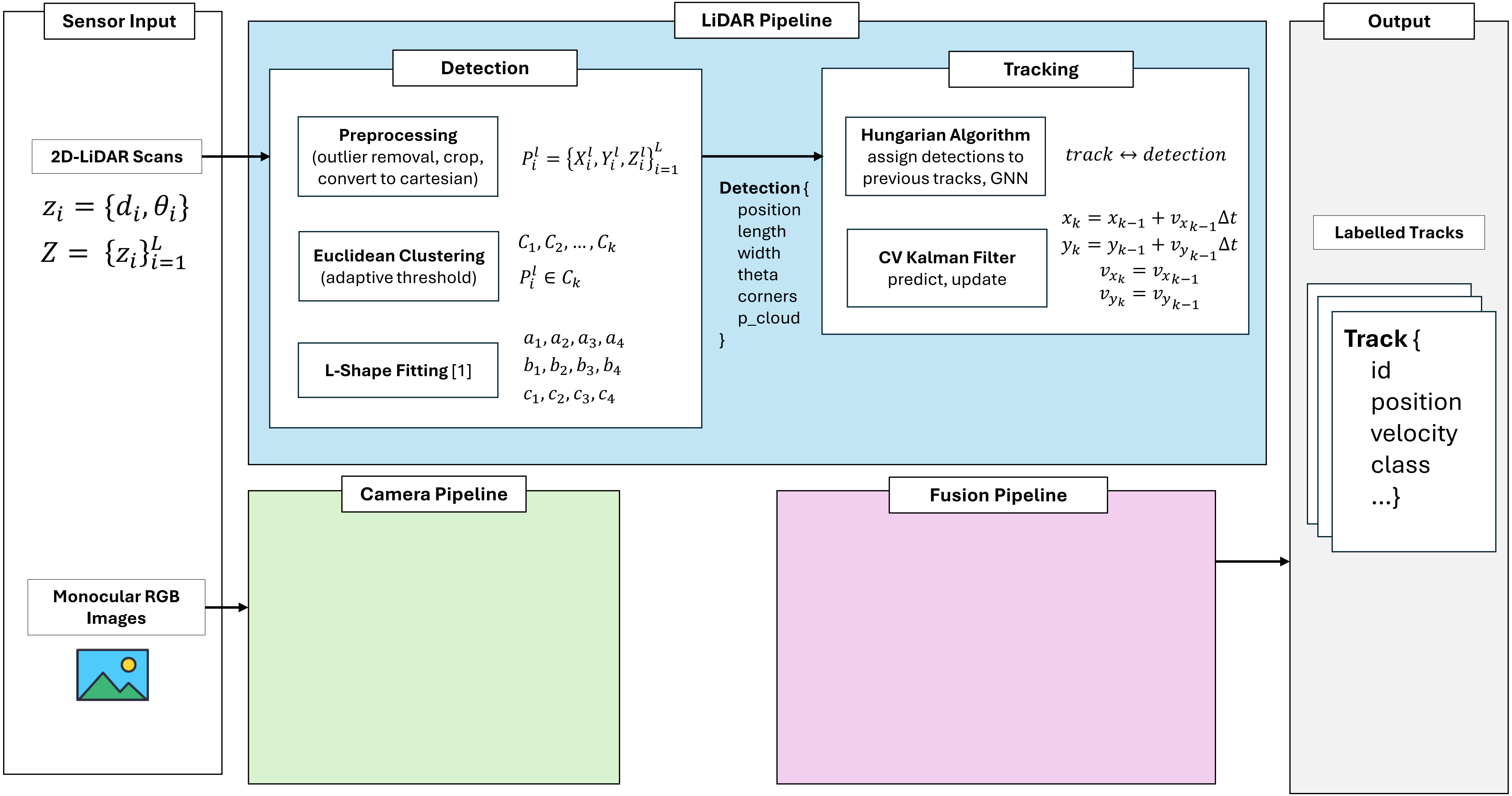
Sensor Input

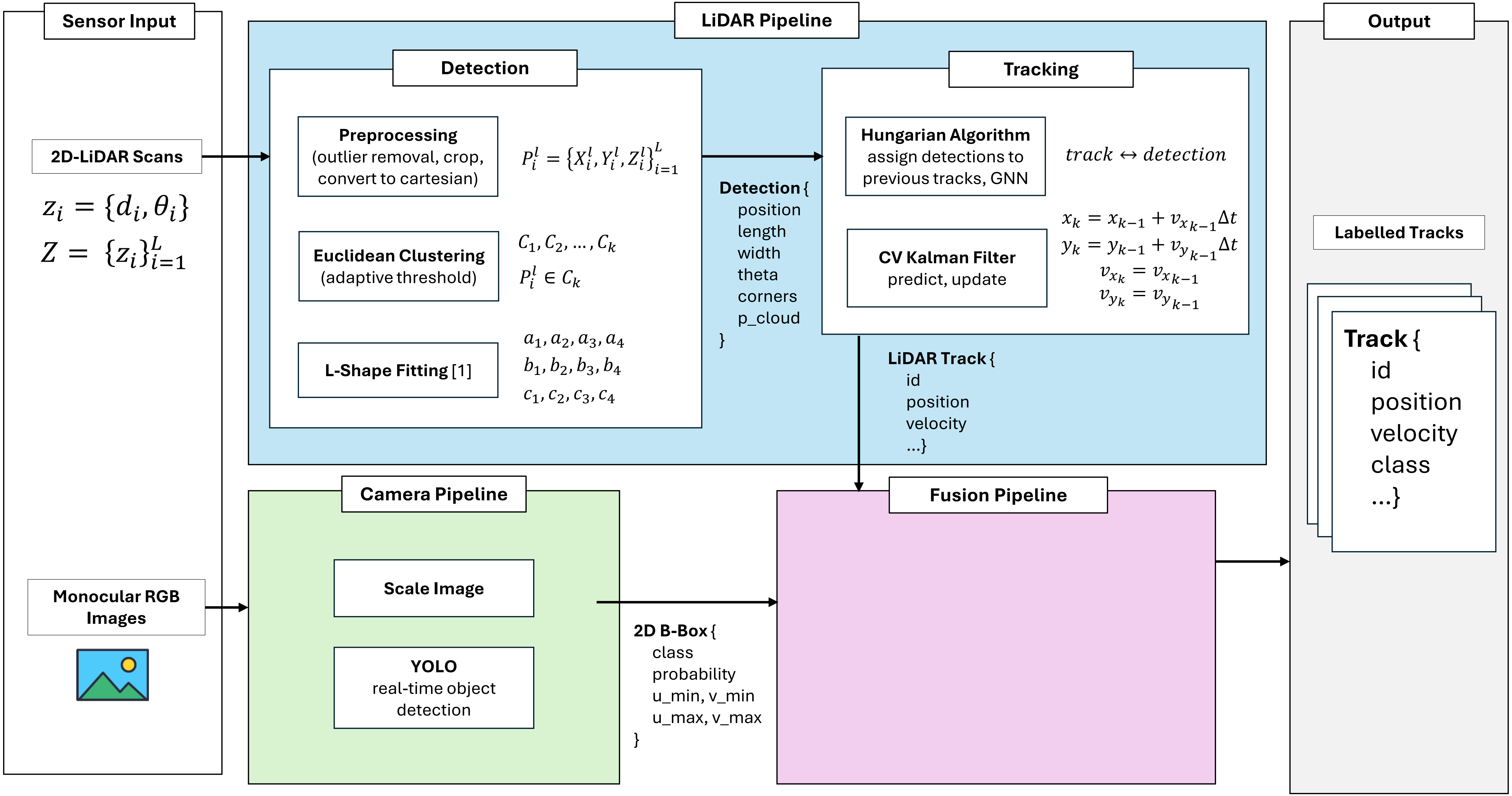
LiDAR Pipeline

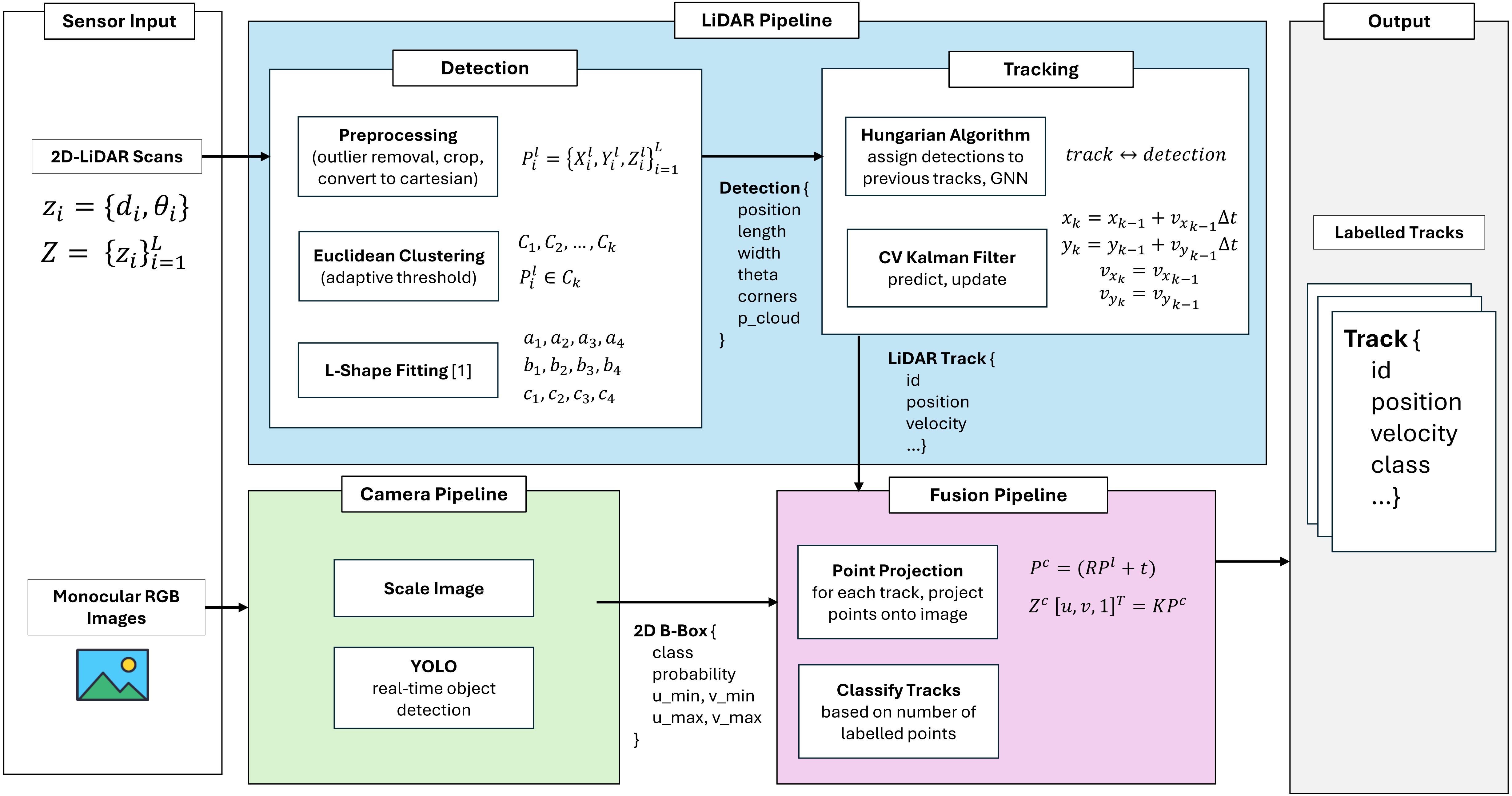
Output

Design Challenge 3.1: Identity switching

Did not consider


Sensor Input
2D-LiDAR Scans
$z_i = \{d_i, \theta_i\}$
$Z = \{z_i\}_{i=1}^{L}$
Monocular RGB Images
LiDAR Pipeline
Detection
Preprocessing
(outlier removal, crop, convert to cartesian)
$P_i^l = \{X_i^l, Y_i^l, Z_i^l\}_{i=1}^{L}$
Euclidean Clustering
(adaptive threshold)
$C_1, C_2, \dots, C_k$
$P_i^l \in C_k$
L-Shape Fitting [1]
a_1, a_2, a_3, a_4
b_1, b_2, b_3, b_4
c_1, c_2, c_3, c_4
Detection {
position
length
width
theta
corners
p_cloud
}
Tracking
Hungarian Algorithm
assign detections to previous tracks, GNN
$track \leftrightarrow detection$
CV Kalman Filter
predict, update
$x_k = x_{k-1} + v_{x_{k-1}} \Delta t$
$y_k = y_{k-1} + v_{y_{k-1}} \Delta t$
$v_{x_k} = v_{x_{k-1}}$
$v_{y_k} = v_{y_{k-1}}$
Camera Pipeline
Fusion Pipeline
Output
Labelled Tracks
Track {
id
position
velocity
class
...}


Sensor Input
2D-LiDAR Scans
$z_i = \{d_i, \theta_i\}$
$Z = \{z_i\}_{i=1}^{L}$
Monocular RGB Images
LiDAR Pipeline
Detection
Preprocessing (outlier removal, crop, convert to cartesian)
$P_i^l = \{X_i^l, Y_i^l, Z_i^l\}_{i=1}^{L}$
Euclidean Clustering (adaptive threshold)
$C_1, C_2, \dots, C_k$
$P_i^l \in C_k$
L-Shape Fitting [1]
a_1, a_2, a_3, a_4
b_1, b_2, b_3, b_4
c_1, c_2, c_3, c_4
Detection { position length width theta corners p_cloud }
Tracking
Hungarian Algorithm assign detections to previous tracks, GNN
$track \leftrightarrow detection$
CV Kalman Filter predict, update
$x_k = x_{k-1} + v_{x_{k-1}} \Delta t$
$y_k = y_{k-1} + v_{y_{k-1}} \Delta t$
$v_{x_k} = v_{x_{k-1}}$
$v_{y_k} = v_{y_{k-1}}$
LiDAR Track { id position velocity ...}
Camera Pipeline
Scale Image
YOLO real-time object detection
2D B-Box { class probability u_min, v_min u_max, v_max }
Fusion Pipeline
Output
Labelled Tracks
Track { id position velocity class ...}



Sensor Input
2D-LiDAR Scans
$z_i = \{d_i, \theta_i\}$
$Z = \{z_i\}_{i=1}^{L}$
Monocular RGB Images
LiDAR Pipeline
Detection
Preprocessing
(outlier removal, crop, convert to cartesian)
$P_i^l = \{X_i^l, Y_i^l, Z_i^l\}_{i=1}^{L}$
Euclidean Clustering
(adaptive threshold)
$C_1, C_2, \ldots, C_k$
$P_i^l \in C_k$
L-Shape Fitting [1]
a_1, a_2, a_3, a_4
b_1, b_2, b_3, b_4
c_1, c_2, c_3, c_4
Detection {
position
length
width
theta
corners
p_cloud
}
Tracking
Hungarian Algorithm
assign detections to previous tracks, GNN
$track \leftrightarrow detection$
CV Kalman Filter
predict, update
$x_k = x_{k-1} + v_{x_{k-1}} \Delta t$
$y_k = y_{k-1} + v_{y_{k-1}} \Delta t$
$v_{x_k} = v_{x_{k-1}}$
$v_{y_k} = v_{y_{k-1}}$
LiDAR Track {
id
position
velocity
...}
Camera Pipeline
Scale Image
YOLO
real-time object detection
2D B-Box {
class
probability
u_min, v_min
u_max, v_max
}
Fusion Pipeline
Point Projection
for each track, project points onto image
$P^c = (RP^l + t)$
$Z^c \, [u, v, 1]^T = KP^c$
Classify Tracks
based on number of labelled points
Output
Labelled Tracks
Track {
id
position
velocity
class
...}

Sensor Input

LiDAR Pipeline

Output

2D-LiDAR Sca

$z_i = \{d_i, \theta$

$Z = \{z_i\}_{i=}^{L}$

Monocular RG
Images

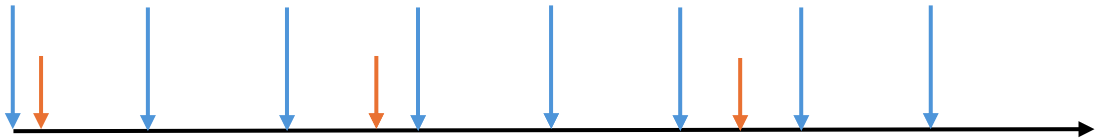
elled Tracks

ack {
id
position
velocity
class
...}

Design Challenge 4: Asynchronous LiDAR and Camera streams

Camera frequency: 15 – 30 fps

LiDAR frequency: 40 Hz



Idea 1: When a track gets assigned a label, it keeps the label until new camera data comes in

Only apply a label to a track if it has been classified consistently

Association back in time: keep circular queue of previous tracks & timestamps, associate to those within 10ms? queue.length() = # of timesteps camera lags behind

Sensor Input

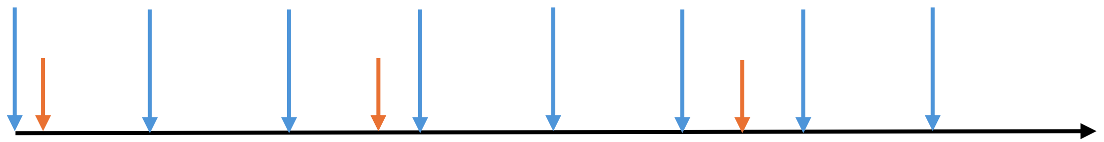
LiDAR Pipeline

Output

Design Challenge 4: Asynchronous LiDAR and Camera streams

Camera frequency: 15 – 30 fps

LiDAR frequency: 40 Hz



Idea 2: Somehow interpolate the tracks

Part 2: Implementation

Implementation

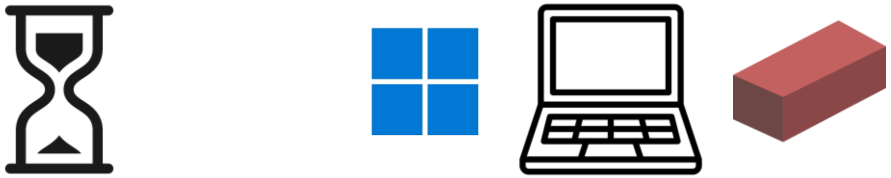
Implementation Challenge 1:

Problem: Learning curve! First time using ROS.

Solution: Looking at documentation, examples, trying and failing!!

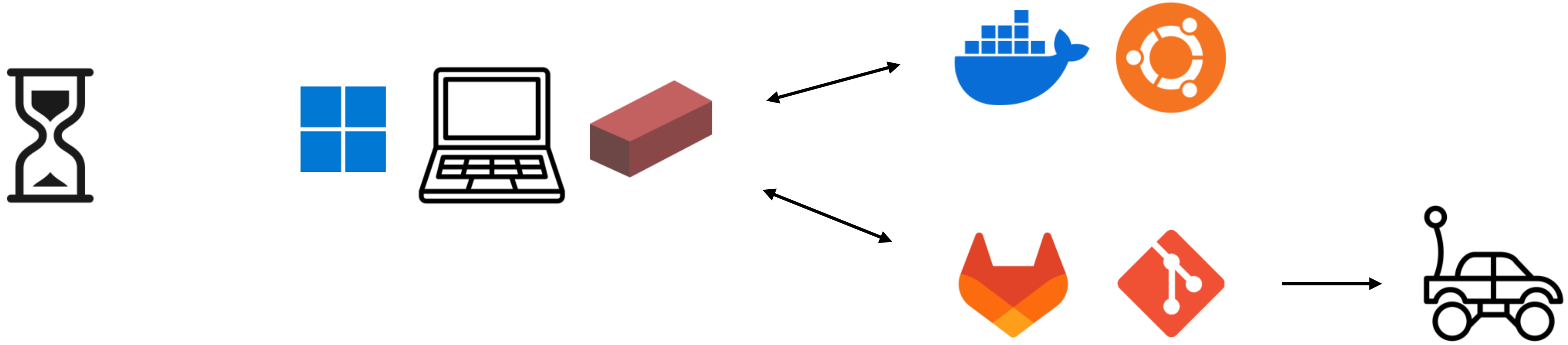
Implementation Challenge 2:

Problem: In the past, students did not have a streamlined way to develop software.



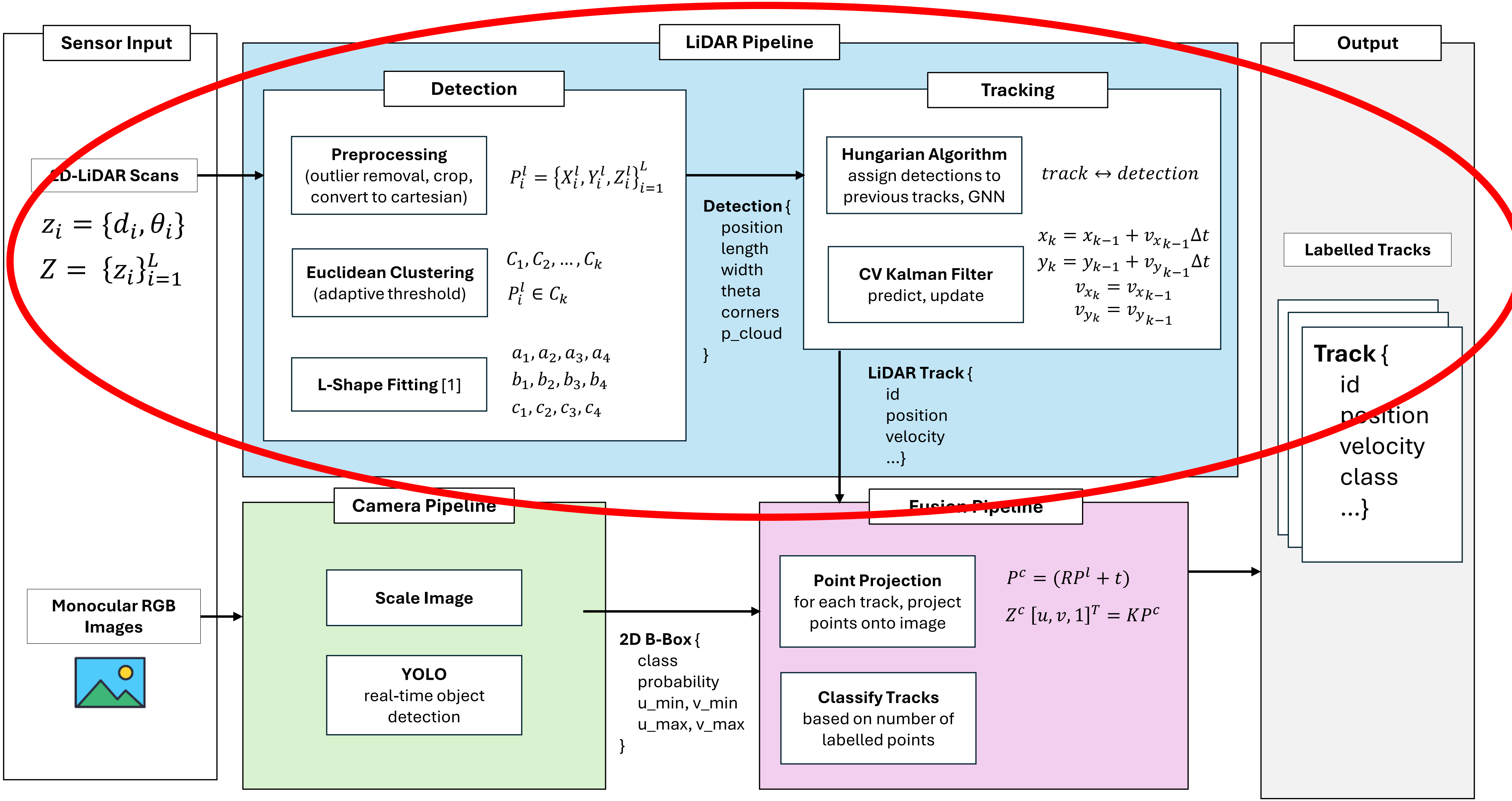
Implementation Challenge 2:

Problem: In the past, students did not have a streamlined way to develop software.



Idea: Using Docker for more streamlined development

Downsides: GPIO trouble, Docker overhead



Sensor Input
D-LiDAR Scans
$z_i = \{d_i, \theta_i\}$
$Z = \{z_i\}_{i=1}^{L}$
Monocular RGB Images
LiDAR Pipeline
Detection
Preprocessing (outlier removal, crop, convert to cartesian)
$P_i^l = \{X_i^l, Y_i^l, Z_i^l\}_{i=1}^{L}$
Euclidean Clustering (adaptive threshold)
$C_1, C_2, \dots, C_k$
$P_i^l \in C_k$
L-Shape Fitting [1]
a_1, a_2, a_3, a_4
b_1, b_2, b_3, b_4
c_1, c_2, c_3, c_4
Detection { position length width theta corners p_cloud }
Tracking
Hungarian Algorithm assign detections to previous tracks, GNN
$track \leftrightarrow detection$
CV Kalman Filter predict, update
$x_k = x_{k-1} + v_{x_{k-1}} \Delta t$
$y_k = y_{k-1} + v_{y_{k-1}} \Delta t$
$v_{x_k} = v_{x_{k-1}}$
$v_{y_k} = v_{y_{k-1}}$
LiDAR Track { id position velocity ...}
Camera Pipeline
Scale Image
YOLO real-time object detection
2D B-Box { class probability u_min, v_min u_max, v_max }
Fusion Pipeline
Point Projection for each track, project points onto image
$P^c = (RP^l + t)$
$Z^c \, [u, v, 1]^T = KP^c$
Classify Tracks based on number of labelled points
Output
Labelled Tracks
Track { id position velocity class ...}

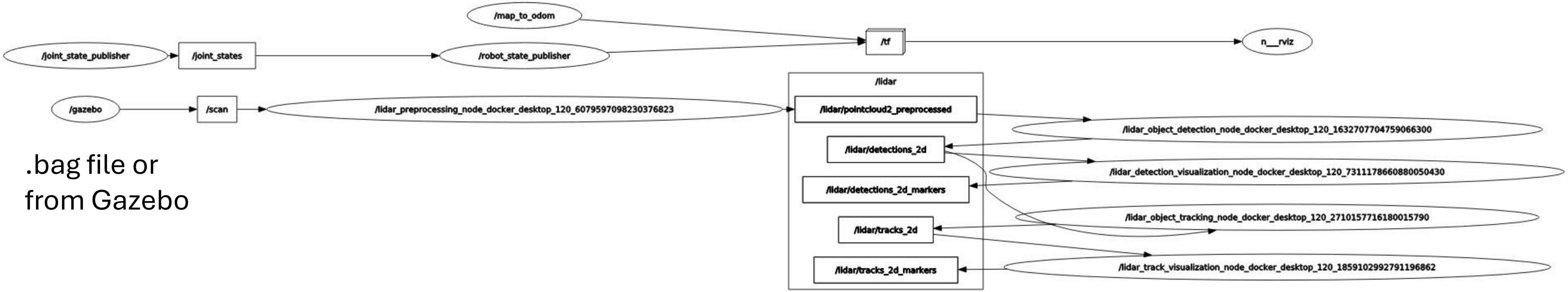
Implementation Challenge 3:

Problem: How do I know it’s working? How do we define “working”?

Implementation Challenge 3:

Problem: How do I know it's working? How do we define "working"?

In this project: Modular development and testing w/ ROS bag & simple Gazebo environment

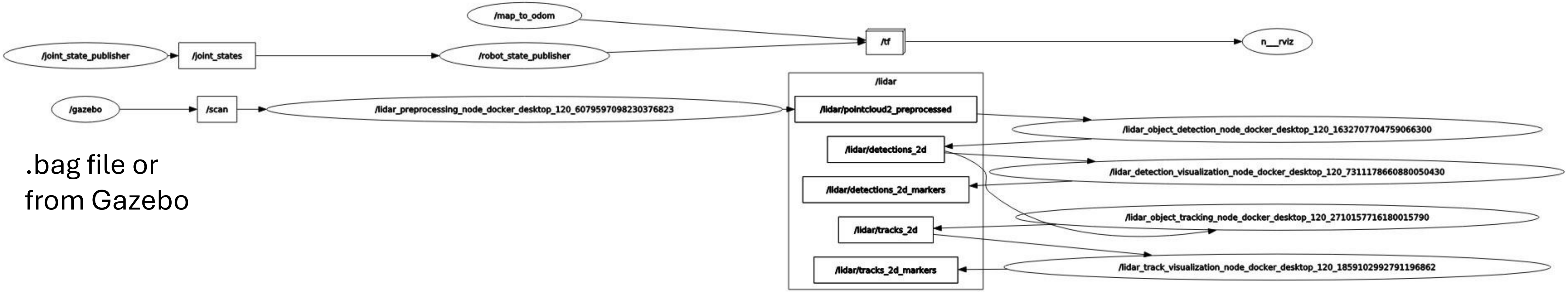


.bag file or
from Gazebo

Implementation Challenge 3:

Problem: How do I know it's working? How do we define "working"?

In this project: Modular development and testing w/ ROS bag & simple Gazebo environment



.bag file or
from Gazebo

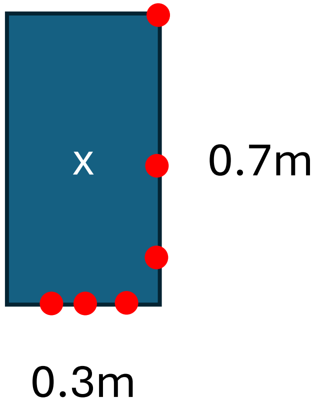
Evaluate? Visually. Not ideal, but it enough for this project scope.

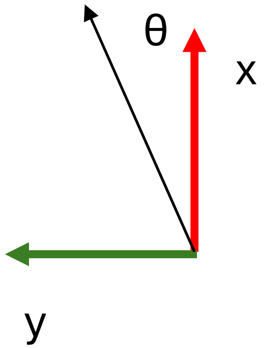
Ideally: Evaluate GOSPA against ground truth + benchmarking

Demo

Implementation Challenge 4:

Problem: Unconsidered behaviour of LShapeFitting algorithm



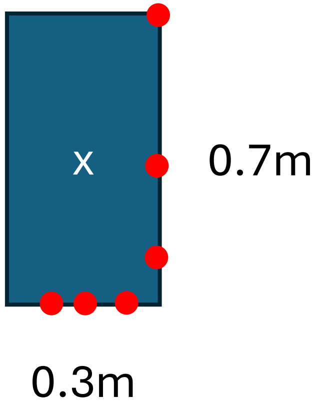


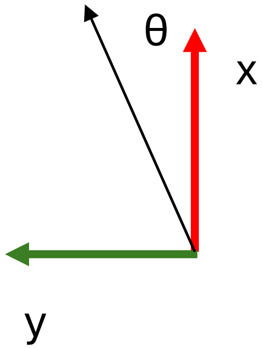
Output from LShapeFitting:

W = 0.3m
L = 0.7m
θ = 0 deg

Implementation Challenge 4:

Problem: Unconsidered behaviour of LShapeFitting algorithm



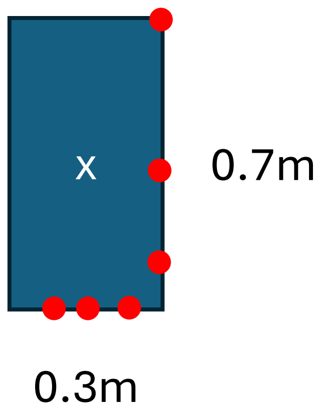


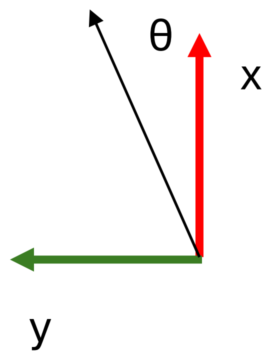
Output from LShapeFitting:

W = 0.3m L = 0.7m θ = 0 deg	**OR**	W = 0.7m L = 0.3m θ = 90 deg	**??**

Implementation Challenge 4:

Problem: Unconsidered behaviour of LShapeFitting algorithm





Output from LShapeFitting:

W = 0.3m
L = 0.7m
θ = 0 deg

OR

W = 0.7m
L = 0.3m
θ = 90 deg

??

UNSOLVED!!

Implementation Challenge 5:

Problem: Finding the extrinsic parameters between the 2D-LiDAR and the monocular camera. The lack of a third dimension makes it hard to find point-pixel correspondences.

Solution: Had some ideas, but ultimately never tried as the camera pipeline was never implemented.

Thank you for listening!

References

- [1] X. Zhang, W. Xu, C. Dong and J. M. Dolan, "Efficient L-shape fitting for vehicle detection using laser scanners," 2017 IEEE Intelligent Vehicles Symposium (IV), Los Angeles, CA, USA, 2017, pp. 54-59, doi: 10.1109/IVS.2017.7995698. keywords: {Feature extraction;Shape;Laser radar;Three-dimensional displays;Clustering algorithms;Vehicle detection;Automobiles},