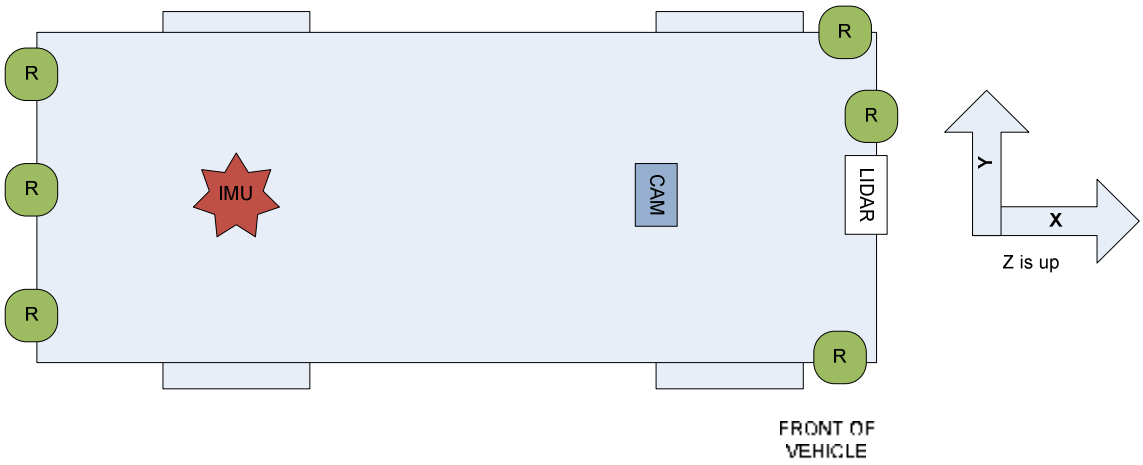
CS664 Final Project Test Data

General Notes:

- *All data is in CSV format.*
- *All packets have a timestamp in seconds as their first element.*
- *Units are meters and angles are radians unless otherwise noted.*
- *All measurements for sensor locations are taken relative to the vehicle's inertial measurement unit (IMU). You will need to perform the appropriate coordinate transforms to align image, radar, and clustering data. The diagram below should be helpful in explaining the vehicle's coordinate system.*



Important Camera Information:

Resolution: 1280x1024
Model: Mono Basler A622f with 4.8mm Wide Angle Pentax Lens
Focal length Conversion Factor: 716.0
FPS: ~16Hz
X offset = .9017m
Y offset = 0.0m
Z offset = 1.7272m
Distance off the ground: 1.74m
YawAngle =-0.23 degrees
Pitch Angle = -8.25 degrees
Roll Angle = 0.0 degrees

Pixels in the camera are square. The camera center is exactly in the center of the image. The camera has simple autogain but it will not respond instantly, as you will notice. All the data in these logs is real and unprocessed, so there will be bad frames, bad timestamps, missing data, false positives, and possibly corrupt data.

Images are encoded as raw windows bitmaps. The filenames represent the image timestamps in seconds. AVI's have been provided to quickly locate your desired images.

Pose Data

High precision position and attitude data from the vehicle. Note there are THREE different logs for this data. Although you can use any of them, it is not recommended that you manually differentiate absolute positions to get relative motion and instead use the relative motion logs.

PoseREL.log

This is a 4x4 matrix which represents the transformation to take a point from time 0 and put it in time T. This means all these transformations originate from the same frame.

Packet Format:

timestamp,[0x0],[0x1], [0x2],[0x3], [1x0],[1x1], [1x2],[1x3], [2x0],[2x1], [2x2],[2x3], [3x0],[3x1], [3x2],[3x3]

Where [axb] is the ath row and bth column of the 4x4 matrix.

Sample C++ code to get the camera's dx, dy, dz, and drx, dry and drz:

```
	//k current
	//i initial
	dpmatrix4 Rki;		//rk,i This is the current xform
	dpmatrix4 Rkmni;	//r k-n,i, this is the last xform
	dpmatrix4 RkmniInv;	//inverse r k-n,i
	dpmatrix4 RkRkmn;	//r k * (r k-n)^-1

	Rki = GetRk(Rk_msg);
	Rkmni = GetRk(Rkmn_msg);
	CalcOptimalInvRkmni(RkmniInv,Rkmni);

	RkRkmn = Rki * RkmniInv;
	double sinDY = RkRkmn(2,0);
	double cosDY = sqrt((RkRkmn(2,1)*RkRkmn(2,1)) +
	(RkRkmn(2,2)*RkRkmn(2,2)));
	double sinDX = ((-1.0f * RkRkmn(2,1)) / cosDY);
	double cosDX = RkRkmn(2,2) / cosDY;
	double sinDZ = ((-1.0f * RkRkmn(1,0)) / cosDY);
	double cosDZ = RkRkmn(0,0) / cosDY;

	//vehicle coordinates, delta angle (yaw rate = drZ / dt)
	vehState.dRY = atan2(sinDY,cosDY);
	vehState.dRX = atan2(sinDX,cosDX);
	vehState.dRZ = atan2(sinDZ,cosDZ);

	//vehicle coordinates, delta pos (forward speed = dX / dt)
	vehState.dX = (RkRkmn(0,0) * RkRkmn(0,3) * -1.0f) + (RkRkmn(1,0)
* RkRkmn(1,3) * -1.0f) + (RkRkmn(2,0) * RkRkmn(2,3) * -1.0f);
	vehState.dY = (RkRkmn(0,1) * RkRkmn(0,3) * -1.0f) + (RkRkmn(1,1)
* RkRkmn(1,3) * -1.0f) + (RkRkmn(2,1) * RkRkmn(2,3) * -1.0f);
	vehState.dZ = (RkRkmn(0,2) * RkRkmn(0,3) * -1.0f) + (RkRkmn(1,2)
* RkRkmn(1,3) * -1.0f) + (RkRkmn(2,2) * RkRkmn(2,3) * -1.0f);
	vehState.dt = dt;
	vehState.timestamp =
(double)(Rk_msg.car_ts_secs+(double)Rk_msg.car_ts_ticks/10000.0f);
	lastMsg = Rk_msg;
```

PoseECEF.log

This is the ECEF location of the vehicle, which is measured at the IMU.

Packet Format:

timestamp,x,y,z

PoseVEHvel.log

This is the velocity solution from the vehicle as well as the absolute YPR.

Packet Format:

timestamp,vx,vy,vz, yaw, pitch, roll

Raw Clustered LIDAR Data

3D Bounding boxes of LIDAR targets.

ID's & Locations:

The clusters are already put in the reference frame of the IMU. You will need to apply coordinate transforms to align with image data.

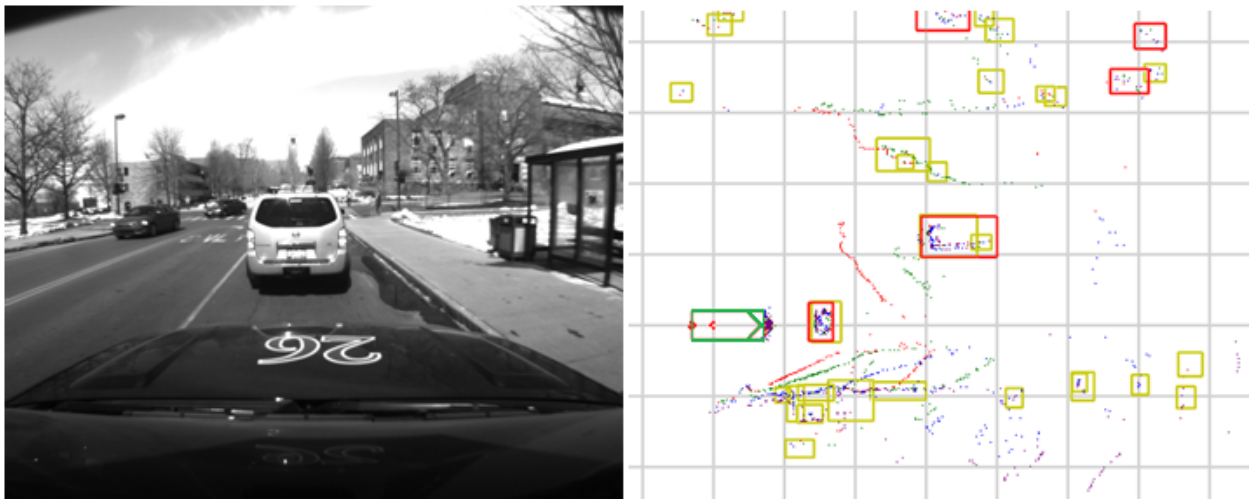
As a note of caution: the clusters generally will only represent the front faces of objects looking at the vehicle. The LIDAR obviously doesn't have the ability to see through the object and really measure its extent. For this reason, you may want to consider averaging the high and low measurements to get a sort of "center" point to start with.

Packet Format:

timestamp, numberClusters, [high x, high y, high z, low x, low y, low z]

the portion in the [] will be repeated for the number of clusters in that particular scan, indicated by numberClusters.

Sample Cluster Data Rendering (log 2, timestamp 2201.3):



This figure depicts what the sample clustering data looks like rendered in a top down view. The Tahoe is the green vehicle. The Red boxes represent clusters that will appear in our logs. The yellow boxes are insignificant clusters that are NOT in the logs.

Image Transformed Cluster Data

Same as the raw clustered data, but projected into image coordinates.

Packet Format:

Timestamp,bb_high.x,bb_high.y,bb_low.x,bb_low.y,leftOccluded,rightOccluded,stable,a,b,c,d,aw,bw,cw,dw

The bb fields are the 3D coordinates of the given cluster's bounding box. They are measured relative to the vehicle's IMU.

Left/Right Occluded indicate if another cluster is in front of this particular cluster. These fields are boolean (True/False).

Stable indicates the cluster should remain close to the same size from frame to frame. This is measured by a seperation from all other points. It is generally a good idea to only use stable clusters are far ranges.

A,b,c,d are coordinates in image space that represent where the obstacle is. These four points form a bounding parallelogram around the obstacle. Note that a "point" is formated like this: **{X=-1236,Y=222}** . Also note that you will frequently get points that are way off the image since the LIDAR has a much wider FOV than the camera.

Delta Pitch Data

This is very simply the derivative of the absolute pitch of the vehicle in ECEF. This may be useful as a signal to know when the car is bouncing.

Packet Format:

Timestamp, Delta Pitch (rad/sec)

Delphi Radars

Tracked targets using millimeter wave radar.
Very narrow FOV (15 degrees) and long range (~100m).

ID's & Locations:

NB: For upside-down radars, you must negate the angles in the measurements!

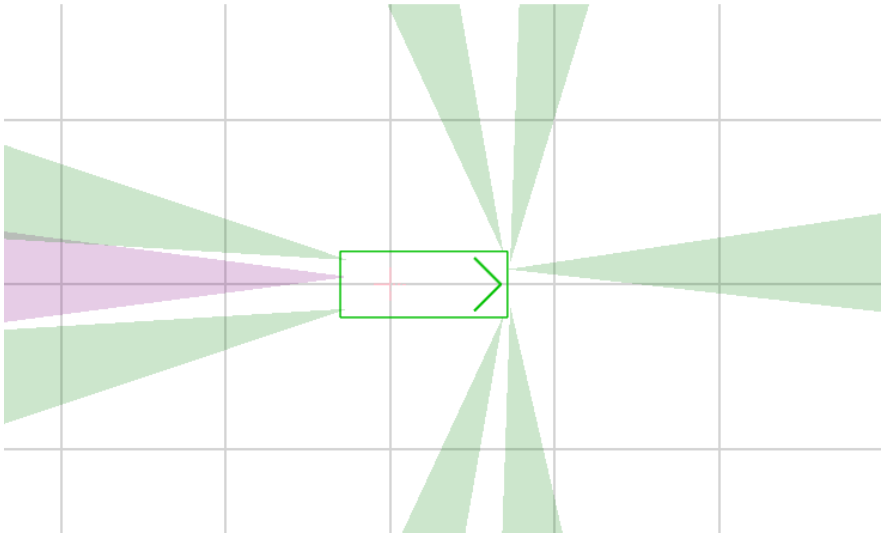
Radar Name	Radar ID	Upside-down	X	Y	YAW (degrees)
RadarZ	7	T	-1.27	.2286	180
RadarB	3	F	3.4417	-.9398	-107.5
RadarC	5	F	3.6449	-.6477	-84.5
RadarD	6	F	3.5687	0.4572	1
RadarE	4	T	3.6649	.6477	80.5
RadarF	2	T	3.4417	.9398	107.5
RadarY	1	F	-1.27	-.7366	-169
RadarX	0	T	-1.27	.7366	169

Packet Format:

timestamp, radarID, numberTracks, [track id, range, angle, range raw, angle raw, power, isMature, rangeRate, rangeRate raw]

the portion in the [] will be repeated for the number of tracks in that particular scan, indicated by numberTracks.

Radar Locations Rendering



The green box is the Tahoe. The cones each represent a single radar's FOV. The arrow in the Tahoe represents the forward direction of the vehicle.