**SLAM Unit A - Part 1**

In this first unit, Unit A, we will learn how to use the motor and the scanner. This is the first part of the first unit, in which we will get acquainted with the notebook and perform first experiments with the robots motor.

**Introduction**

Problem buying such a thing so we build our own as you can see here. We have two systems. One is the Google Self driving car and this is our own robotic system. So what's the similarities on top? You have, as I mentioned, this lighter. And we do have a lighter as. So instead of this bulky Velodyne 1,000,000 points per second 70,000 US dollars scanner, we have this small lightweight hokuyo scanner on top of our robot. And we also have a different DR mechanism. So here you have normal tires whereas we do have a caterpillar system with Caterpillar tracks. Well, an otherwise well that's just a car. So it has somewhere here in there. As a drivetrain driving here, the probably the rear access. We have the two right, so we have two of those tracks here. We have two motors here driving the left and the right track. And so the vehicle moves forward. Our device is actually much, much cheaper than the original Google Self driving car, while on the other hand there is some little drawbacks, one of them being this car. There's actually a self driving car. There is. This car is currently driven by Daniel. So here's Daniel. He designed and constructed this device, and he also has built the control software which is able to control the movements of the small robot, but which is also able to get the measurements. Of this laser scanner device in real time. So let's have a closer look at the system. We do have the lighter up here. The lighter has an access here, and the lighter is shooting out its race in that direction, so horizontal, so parallel to the ground. And there's some area behind here which is not. Covered by a lighter, right? So that's a dead area which where the lasers cannot can't see anything. And we do have our caterpillar tracks here. These are the driving motors. Here and the other one you can. This is an extra battery and this is. The controller the drives here the two motors and now the controller links wirelessly via Bluetooth and also the lighter data is read out via serial interface right? It goes into this. Bluetooth device here, so that also is sent via Bluetooth. This is our robotic car in his natural environment. What you can see here is our arena. And so we placed some obstacles. These are actually not meant as obstacles. Those are. Our landmarks. So later on we will try to find those landmarks in our scan. To find out algorithms to control the movement of the robot. So how is all that set up? So all that is here. So there's no wiring, it's all Bluetooth. And here's the control computer. And with this control computer you can tell the robot to move forward and you get back the signals of the motors, the motor count and you also get back the measurements of the lighter scanner. Here you have another view. And here on the screen you can see now at live view of the scan that the robot does. So here's the top view of everything. The robot is in the left.

## Motor control

They ran our robot through our arena. Let's look at motor control. So this is our robot. Here's our lighter the scanner, which builds scan somewhere in that range. And we'll also have an invisible area here, and these are two motors, and the two motors they drive these access, meaning if they go into certain speed, the robot will also go at this speed. And if one of them goes faster then the robot will somehow make it true. Now these motors they do have a wheel encoders, meaning there is encoders that count the number of turns. Actually not single terms, but they count many ticks for a single revolution. This tick count is sent to our control computer. Our control computer just locks all those values here. Is one of those log files. And this looked fun. Is also in the set of files that you have for the current unit. So this file is called Robert four underscore motors dot text. And we made. Some code at the start of each line. So here you're writing M for motor motor information. And then all the data. So from this data actually don't need very much. So this is a time stamp. So this means that millisecond 200 for the motor readings were as follows. This number here is the encoder value. Over the last motor and we don't need these three values. And this is for the right motor you'll encounter. And there's a third motor which would be this one but which we are not using. So all the other welders are not interest if they want to read this file, just keep in mind. What we need is we have to check if this is an M so this will be the motor readings. We could use the milliseconds here. Then we need this value and that value meaning if we count. This is 012345 and six. So we are actually just interested in the valley too and in the value 6. So also if you look at this. And collar bones. They do not start at 0, so they start at a certain number and then when the robot moves forward they start to increment at a certain point in time. So remember left. That's the second column and right that's the 6th column we are interested in. So now let's try to read out the motor file. So in this file we have those lines. Which are M, then at timestamp, then the position left and three other values which we are not interested in, and then the position rate voting ticks. So first of all let's open the file. But far less robots, robot, four motors, dot text, and so we open it. First of all, let's just print out all the values. So we say. For line in file. Print the line, then we run it. And you can see it brings out those lines that we've just seen on the last slide. So now let's read out the left and the right motor ticks. So start by making. A left list. Which is empty and right, which is empty. And then I have to split up the line into its columns, so I say. SP is a split up of the line. In fact, let's just print. SP to see what happens. Split command splits up the string in parts and it will have the M column #0 column number 123 and so on, and so we can just grab the columns we'd like to have. So we. Take the left list, but actually we're interested in the in texture and not in the stream. So we converted to an internship and we happened to the right list, the 6th column. Let's try this. But we can't see anything, so let's just ask for left list. So this is all values of the left encoder. So now that we have this, let's just block this. In order to plot, we import pilot and then we just plot both lists, both left list. And plot right list. Let's run it. So that's the outcome. So these are the values of the two incremental encoders. So the left one starts at a little bit more than 20,000, the right one starts at more than 15,000, and then the robot starts to move. And then just increment and we can see that the left phone starts at a higher value, but in the end it ends up in a lower value. So this means the ride hailing coder has more cakes. Along the entire trajectory and the last one, so the robot does the leftover. So the absolute tick numbers are not really so meaningful, and so it's hard to interpret the drawings that we just made. So let us build the difference of pigs. And now we could just program this, but I'll show you something else. So there is a clause which is in the package that you downloaded for this course. Which is called LEGO lock file. You can import that, and the purpose of this class is to import all different kinds of records that will produce throughout this class. So in order to read the file, you can just lock file equals legal lock file. And then just read it. And if you run that now we have this class. And for example, we can ask for the motor tics. And those ticks are the differences from one time step to the next one. So let's just print out the 1st 20 more tips. And here they are. As you can see in the beginning, the robot does not move, and then after a while it starts to move, it starts to accelerate, it gets faster and then here's a little bit of difference. So the left motor. Those wanting faster than the ride motor, then we also having something else here we do have some time like and you do this time like we have 00. Well you spread across our measurements if I make that. Longer, you will see that occurs more often. See, at about every five movements or so, we have a 00. Now let's block those values in order to see what's going on. So I'm adding. The import from pilot and then it is as simple as just adding the blood command down here. Let's see what happens. Nanda spoke. We can see the incremental values for the motor encoders and we see those spikes going down to zero, which is not what we actually want, but we can also see. The robot drives straight. And then the right motor. Gets faster, whereas the left motor gets slower. So the robot mixture, then it goes straight again and so on. So you also can use this. To have a closer look.

## The motion model

But what we want to know is whether Robert is. So we need to know how the robot takes translate into your. Movement of the robot. So we need a motion model. Let's have a look at the robot once again. This is the robot as seen from above. And these are the caterpillar tracks of the robot which may move. The different speeds. So let us just assume for a moment that these were now tracks. But these were just wheels that are placed here and here, and so would be assumed that the robot has some center like here. And some axis or go through here, whereas the left and right motor will turn those fields at different speeds. So then our motion model is as follows. Robert has left field and the robot has a right wheel and if the robot moves and say the right wheel moves a little bit faster than the left wheel. Then they're over to make some turn so it will look like that. And the robot will turn around point say this point is here, so in the interview will be here and here. So the movement of the left field, this curve segment left length of L and the right field has length of R. And we need to know alpha. That's the angle we want to find out, and we also do have a radius. And we also know the width of the vehicle, so the with the distance between the two wheels or tracks of the river, so having a look at that. We have a situation for the right and for the left field, but for the right heel it looks like that this is our angle alpha. This is our. And down here. That is R plus the width of the vehicle. So from that you can set up the equation right equals alpha taken in radiance times R + W with. And for the left wheel left, the situation is a little bit shorter here like that it's alpha select and that's R so for the left wheel the equation is left equals alpha times R. So if you subtract that from each other, then we get r -, L. Equals alpha times the rest of the vehicle and so they're unknown are is dropped from resignation. So from that we obtain that alpha, the angle we're looking for, equals R -, L divided by division. And now if we put that into this equation here. We also find out that this unknown R that is L divided by alpha. So what we can see here is we have played two equations. The first one is from the movement of the right and left wheel. And the known width of the vehicle we obtain an angle alpha and having these angle alpha we can actually put it in here right now the left length and we know alpha so we can compute R so what we know so far is. Alpha and R and we also have to note that alpha is not allowed to be zero, because then we would have a division by zero. And that makes sense because if I -, L is 0. Then there would actually go straight, and this point here would be in infinity. Now all over to somewhere in the real world. Let's say this is the real. This is the right wheel. This is the axis. So the robot will move in that direction. I'd say around a point as we just discussed. So this is the point, the center around which it moves and we'll move up here. This will make this curve segment, this will make that curve segment. We'll end up here. So this is the moving direction. That's the heading, and we'll say the heading angle. It will use theta. To describe the heading angle then this vector here will be cosine theta, sine theta and we now construct a vector that is rectangular to this heading direction and this will be. Sign. Minus cosine theta. You have to exchange the two terms and you have after exchange and you have to make a minus depending on if you turn that right by 90 degrees. You have to make the minus down here if you would. And last year you would make the miners up here. So now where does this point say this is the center or return say this here is the position of the Robert P Now the distance between here and here. We know that is R plus half of the width of the robot, so overall we get the center. Is that the position here minus the distance times this vector? So after we drive up here. Where will be this position? What we'll find this position? By also constructing this vector in that direction. And adding our plus the half of the 5th to here in that direction. This direction is the same as that direction after it has been turned. Now we determined the turn is alpha. So this direction here is actually same as that but now with Alpha added. So would be obtained the new position of our robot P Prime which is the center. Plus again our plus half of the width times the sine and cosine, but this time of theta. Plus alpha. And also we get the new heading which points in that direction up here and the new heading will be just the old heading Plus alpha. Now since we want to keep that within 2π, it just. So a model division to pi. Alright, and remember in Python that will be Modelo. Two times part. Let's wrap that up. So if you are given the last position and orientation of our robot, which is XY and theta and we're also given the view counts of the left and right. Yeah. And we also know the width of the vehicle, which is a calibration parameter, and we're looking for the new position. So we first compute our angle alpha, the turn angle. Which is R -, L divided by the width. From that we can compute the radius, which is L divided by alpha. From that we can compute the center. Which is still position minus R plus half of the width of the robot times this vector which uses the old Henny. Now we can compute the new heading the new heading instilled. Setting plus the current alpha model to pi and from that we can compute the new position. Which is the center plus again the radius plus half of the width times the sine and minus cosine of the new theta. And this is all for the case if alpha. Is not zero, or equivalently if R is not equal to L Now the second case if R = L, That's actually much much simpler. So. Moreover, is here. It has this heading and they've already equal to L. It chest moves straight so it ends up here. It is easy to find out the formulas are theta. Prime is just theta because. We didn't change our heading. Where's our ex prime? That's the old X plus the distance we drove Albright L here. Now since L is equal to R, you can also write R times. The cosine of theta and Y is the same using the sign, so this is the second set of formulas which we do need when the robot goes straight. There's two more things you need to know. First of all, in the formulas, you need the width of the robot. So what is the width? It turns out that's actually not so easy to determine, because as you here, our robot doesn't have this. Mathematical wheels that they used to figure out formulas. Rather it has this caterpillar tracks. We assume that it's some mathematical wheels like that. What I did is I took a ruler and measured. Distance between the center of the two caterpillar tracks. Using that method, I figured out the width is 150 millimeters. So in all our computations will be based on. So as you remember, we plotted the ticks of the motors, but the ticks, that's what the wheel encoders deliver. The ticks is not the millimeters that we drove. So you need a factor so for every tick. That the building code accounts the robot moves a little bit. In my opinion the factor is .349. So I figured this out by lighting the robot drive straight. After driving for a certain number of picks, I measured the distance that it drills, so it's awkward. 349 meaning one kick equals 0.349 millimeters. In the real world.

Implement this motion model and for that I have prepared some code for you. So the code consists of a main function and the filter step and you will have to implement this filter step. Filter step against the old pose, which consists of XY and Eddie. And against the motor takes left and right and it has to implement the motion equations for the two cases. First case is the motor tics, left and right are the same. In that case the robot just drives straight. And if not, the robot drives along a curve segment, and you'll have to implement that case too. And these are the formulas we just discussed, and I prepare the main function for you. There's a constant we just discussed. It takes 2 millimeter conversion factor. Another constant is the whistle to drag in millimeters and the main function does all the opening of the file and the processing for all the motor ticks. So here. It opens the lock file. It reads in the motor tics. Then it starts to give you post. So since I don't know the post, I just say it is at X0Y0 and with a heading of zero. And then I start to construct the list of the filter positions for all ticks. In the motor tics. I just called this filter step function the one up here, which you'll have to implement. The filter step function takes the pose takes takes and computes new posts, and then the new post is just appended to this list. But in the end I do two things. I print out all the positions and orientations and also in addition to a plot. So let's have a look at what the program does after you implemented the filter step function. So just run it. And this is what happens. The program prints along list of XY adding values. So if I scroll back. These are the first values at the start of the experiment and the robot standstill. So it is at 00 with adding zero and at the beginning the motors are not turning and for the first. 12345678910111213 for the first thirty timestamps. The robot stands still and then starts to accelerate and it goes straight. Since the heading is along the X axis, it goes into X direction and it accelerates and then as you remember the third value here the left and right things are a little bit different, so it starts to turn a little bit than the heading here switches. To value something 2π. So when you implement this. Make sure that you have those 13 zeros here and then same values as you can see here. But the program does not only output those values, it also draws figure. So here's the figure it draws and you can see. Starts at 00, was heading along the X axis. Cedar Point zero. Then it moves and does a left turn until you see for the first time you see the trajectory that the robot is moving. Ending arena. So now try to implement this. Had to look at the graphics. And make sure that you get the correct values as they are shown here.

## Modifying the motion model and generating file output

So now that you programmed the position tracking, I'll ask you for three more modifications. So first of all, as you remember, we had some colored cardboard on top of our. Laser scanner and this laser scanner was then tracked by our video tracker. And you probably remember this red circle which tried to follow the laser scanner in the real world. So as you can see. There is more than one coordinate system involved, so we try to track this point here, whereas the robot turns around a point that is somewhere here. So there's a difference between the robot coordinate system and the track coordinate system, and this is often the case. So you often have for every sensor you do have coordinate system, and for the robot as such you would have. Another coordinate system, which often is termed the body coordinate system, is the lighter coordinate system. So we have the displacement, and the first thing I'll ask you is to integrate this displacement. In this case it means you should output. The point of the lighter whereas for your motion equations, the coordinate system of the robot is important. So what you can see here, it's really easy to do so because if the robot has this heading then you just have to subtract this displacement here and then after you did your forward motion equations. You have to add up again this displacement vector which then points to another direction in the direction of the new heading. I assumed that the robots coordinate system. It's somewhere here and I took a ruler and measured this and I think it's something like 30 millimeters. Now the second modification I want is a different starting point, so if you remember the video. From the beginning that was our arena and the robot went from the bottom left. Now unfortunately that video was turned by 180 degrees. So in reality now coordinate system the robot starts here and goes down here. And then dust those circles. And the third modification I want you to do is output to file. So instead of directly printing to the console and making this figure using math. I wanted to output all positions into an asking file. You'll have to write this second implementation for the tracking of the robot. And you have to add the scanner displacement, the difference. Outpost and the output of the result to a file. And so as you see, we still have the same function as you did in your last implementation. It's only the case that you have to integrate into this function. Now this kind of displacement, and that's all this kind of displacement is defined down here in the main function. Otherwise the ticks and the robot with are still the same. And there's a new start post which I defined here with the new. X, Y and in your head, as you can see here I opened this motors file. I do the filtering just as in the last solution and at the end here's the output to file instead of the plot. So I open a file and for every post that is generated here it just output an F which is the code that we use for filter position and then the XY and heading. This X1 heading this is exactly 3 values that you should produce for every post. So and after you run this code it will produce a text file which is called poses from text. Meaning you can get all the posts of the robot using just the tick counts from the motors and this file will have 278 lines and you should check that you produced that number of lines. And it begins as follows. So here is the F it always is in F for filter precision. And these are the values that are set in the program as the start post and as we had that before, you have first this 13. Positions where the robot stayed in the same place and then it starts to move and so you should verify that your program gives you exactly the same numbers. And let's have a look at the end of the file. So at the end of the file the position. Should be 329543 and the heading should be 4.78 and you should verify that you get the same result and there's something really cool in the set files that you downloaded, so after you produced. This file with all the positions. You can use this log file viewer dot pi to have a look at it. So locate the file in the set of files you downloaded for this unit and then started. Either from the console or via idle or just by double clicking on it in the explorer. So after you start the log file viewer, it will open up a file selector like that and you can switch to text files in order to only view this type of files and then select the posts from text. And that is the file that we just produced. And after you opened that you can see the trajectory, all the points that you just produced. And you can now travel along this trajectory with your virtual robot starting at position 0. And going to position 270 cents. So this is the 278 positions that we have here and you can see the position as well as the heading at that point where the heading is shown as a line. Pointing in the direction of the robot. You can also see the numerical values that you produce, meaning the XY and the heading. And if you want you can load additional files, for example. The motor information that you're used to compute this trajectory, that's the robot motors file, or in that case that can be visualized. So you'll see the motor values down here, whereas it doesn't change the display of the trajectory here.

# Part 3

## Sensor data: laser scanner (LiDAR) data and its derivative

why do we need sensor data? So if you now look at our solution, but you could be happy with it. So it's pretty smooth but we don't know if it is correct. So in order to find that out, let us load the reference trajectory and this is called robot. For reference Dot text now let's look at this. So in red you now have the reference trajectory that was obtained from tracking the robot via the overhead camera. So let's go back to the start. In the beginning reference point and the trajectory that you computed from the motor takes, they are the same. That's because I have given the start position which I just grabbed from the reference to check 3. And as we move along, things are pretty good as long as we go straight. But I have a certain moment. We see that as the robot starts to turn there is a deviation. It seems that curve it takes has two small radius and this leads to a deviation and this goes on. So the next curve. It's also too small rookies so and as you see everything is banned here and after a while our robot is in a completely wrong position. So as we remember I have measured the width of the robot has given you the width of 150 millimeters. So. But it was not so quite clear where the middle of this robot tracks where. So one thing you can do in your. Filter mode of file exercise. You can change the robot to Smith, so let's change that to, say, 160 millimeters and then just run. Now we can go back to our visualization and justice press reload all. And as you could see right now, the trajectory moved a little bit. It's better now. So the terms it takes, they are still too narrow, but it's better. So let's try another value. Let's try, say 173. Go back and reload. Let's see now. We are pretty good now. So by setting a good parameter value you obtained trajectory that is much better. But still this should leave you somehow worried because the width of the robot. It's not just assumed to be 173, but we never mentioned that. It just seems that this is a good value. So what we do here is actually a calibration of the values. But the problem with this is it might work now very for our condo robot and for our current ground. So. But if you use another ground, we still may be off because we will have a different clip of the robot. On the ground. So we need another solution and this will be the measurement of our landmarks by using our lighter. Now let's have a look at the lighter data. So open up the log file viewer and in your directory. For this unit locate the robot for scan text. So this file contains all the scan data from our robots slider. Just open this and on the right hand side you see the robots coordinate system. So the things you saw so far were on the left hand side of our viewer which is in world coordinates, and on the right hand side we have to see shown in the robots coordinate system. Now let's move the slider and you'll see that is the lighter. Data the robot sees. That's the robot travels through the arena. It measures those points. Robot moves forward and forward is his X direction, and then the scanner scans from here to here. All those beams. It's a total of 660 beams. And then behind the robot. There's a zone where it can't see anything. Now what are those spikes? You'll remember those landmarks in the scene. Every landmark leads to one of those spikes here. There's the landmark is here, then the laser rays that go along here. They hit this landmark, the landmark casts a shadow. So currently if you look here, we have 12345 landmarks overall and as I go through the scene this varies so in the starting position. I do have 123456 landmarks and remember these were all the landmarks that we had in our scene. As I move a little bit, for example here to step 70, you see one of the landmarks disappears in the shadow of the other. And so we only have five of them. And this goes on. When we are back here, we only see two landmarks. We go around here at this position, we only see one landmark. Then we go back here. Again, here's a shadow landmark. Now it moves out of the shadow and so on. This is our entire scans scene from the robots perspective. Nice all the other data, the scan data is just a text file and so if you look into the directory for this unit, you will find the robot for scan dot text file which which just opened in our log file viewer. So this file. Contains one line per scan position. So you can see. Here's the first line that starts with. Works around here, goes on and on. So it starts with an S it goes on and on, and then here's the next line. So in between all those values are measurement values. It goes on like that and the entire file. This is our 278 scans, each one having 660 range measurements. So how is one record? One scan line store, let's always be do have a code that starts the line, this case it is an S or scan data. And then there's a timestamp 350 milliseconds, and there's count, meaning there's. 660 values which followed now and then. This is just all the values. So this is 660 entries and those values are the best values of the scanner. Now let's plot the scan data in the file to download. There's a python plot can file and it's very short and simple. So it imports from pilot for plotting, than it imports everything from legal robot, then it loads the log file. In that case it loads the scan data. And then it just plots the scan data. Now log file dots scan data means this is the list of all scans. So if you take the 7th scan here, we just have to index that by a seven. Now let's have a look. And this is how it looks like. This is our 660 scan values down here. That's the indices. And this is the death that the scanner delivers. So you see the robot stands close to wall of his arena and so this. Causes the death values going up here and then there's one of those spikes, which means here's the landmark. We can see more of those bikes. So there's 123456, so all of our six landmarks. Are visible in this can. Now let's have a look at another scan. And I'll switch to scan #8. Let's run it, and the outcome is this. So since the robot didn't move very much, we still have our 123456 spikes, but we also have an artifact here. Let's you look at that. You can see this is a spike going down, so it's an error in the measurement and it's going down to the value of zero. Well, almost. Let's have a look. It's not really 0. It's something like 50. So in order to filter out the bad values, we will assume a threshold of 20, meaning we will not use any measurement value that is closer than two centimeters to the robot scanner. Let's think about strategy to find the cylinders in the skin. Now. As you have seen, the scan data looks like that you have a certain debt and then there's a cylinder in the foreground, which means there will be a death jump in our scan data. It's cancer crystal cylinder, and then it will jump back to hit the background wall. In the real world, this will look as follows. There'll be a cylinder, there will be some wall in the background, and the scanner now will shoot its race. It will hit the background, then we'll start to hit the cylinder. These are these fellows here. And after it went across the cylinder, it will go back here and go on hitting the wall. So this will somehow go on, go around the corner here maybe, and then it will hit. Another cylinder and so on. So now what's our strategy to find those spikes? As you can see, there's a strong negative slope at the beginning of the cylinder and a strong positive slope at the end of the cylinder. Let's think about how the. Derivatives looks like it's going up here, so it will be passed here. Then there's this strong negative P so the derivative will be like that and that's flight actually. So it will be zero, it will be strong positive peak here and then it'll rise here for a little bit. Will switch to a slight negative slope. Then again, there will be this strong negative peak and this strong positive peak. So our strategy will be to set up a threshold just say whenever the derivative is larger than the. Threshold. Then we will detect this as a falling or rising edge. In this case, because it is strongly negative, this will be a falling edge or the left edge of a cylinder. This will be the right edge of the cylinder. And again. Here we will have the left edge and the right edge. So how do we determine the derivative now from image processing we can do this using discrete masks derivative. The discrete position I that is approximately. The function of position I + 1 minus the function that I divided by the step, which is in this case one that's also termed the difference quotient. But this function introduces a phase shift, so we'll use another one. And since there is now I plus form and I minus form, the difference in step is two, I have to divide by two. So now let's implement this and I've prepared this file scan derivative question for you. Let's have a look at this. So there's the main function down here. It sets up a constant which is 20, which is the mWhat scanner? Now let's think about strategy to fight the cylinders in the skin. Now, as you have seen, the scan data looks like that. Do you have a certain depth? And then there's a cylinder in the foreground, which means there will be a death jump. In our scan data, it's scans across the cylinder and then it will jump back to hit the background wall. In the real world this will look as follows. There'll be a cylinder and will be some wall in the background and the scanner now. So should its race, it will hit the background. Then we'll start to hit the cylinder. These are these fellows here, and after it went across the cylinder, it will go back here and go on hitting the wall. So this will somehow go on. Around the corner here maybe. And then it will hit another cylinder and so on. So now what's our strategy to find those spikes? As we can see, there's a strong negative slope at the beginning of the cylinder and a strong positive. So at the end of the cylinder, let's think about how the derivatives looks like it's going up here. So it will be positive. Then there's this strong negative peak, so the derivative will be like that and that's flight actually. So it was zero, it will be a strong positive. Speak here and then it will rise here for a little bit. It will switch to a slight negative slope. Then again there will be this strong negative peak and this strong positive peak. So our strategy will be to set up a threshold. Just say whenever the derivative is larger than the threshold, then we will detect this as a falling or rising edge. So in this case, because it is strongly negative, this will be a falling edge or the left edge of a cylinder. This will be the right edge of the cylinder, and again here we will have the left edge and the right edge. So how do we determine the derivative? Now from image processing we can do this using discrete masks, the derivative of the discrete. Position i. That is approximately the function at position I + 1 minus the function that I divided by the step, which is in this case one. Let's also termed the difference quotient. But this function introduces a phase shift, so we'll use another one. And since there is now I + 1 and I -, 1, the difference in step is two, I have to divide by two. So now let's implement this and I prepared this file scan derivative question for you. Let's have a look at this. So there's the main function down here. It sets up a constant which is 20, which is the minimum distance that we will assume to be a valid measurement value. So any distances below 20 millimeters. Are considered to be an error and it just loads the log file. It picks out a scan like #7 and you're encouraged to try out your implementation with different scan numbers and just assigns the log files can. To scan and then it computes the derivative using the function there you'll have to implement up here and it just plots all the values. It plots the scan as well as the derivatives and. Let's have a look at this compute derivative function so it gets the scan. So the scan will be a list of values of depth values. So let's say this will be something like 100 and 110 and so on. And in the end the last value might be 550, the first item. Will have index 0 second will have index one. The last one was our scanner. Will be index number 659 because there will be 600 sixties current values overall in our scan. Remember I told you to compute the derivative using this formula. So if I is zero, you will access element minus one and plus one. And of course you don't want to do that. Python will let you do that. That it will give you for the index minus one this value here assuming that the list is cyclic and this is not what we want. So the solution here is I start with index one and I run until index the length of this. And minus one, which means the last index is actually the length of the skin minus two. So this will be the last index that is being accessed, and this will be the first index. So now since we only compute. How these values here we will have a total of 658 values in our final list. But I want the final list to have exactly the same length as the original list. So what I do here is I start by adding a zero and I. 10 to 0 in the end and you will have to replace this append here by appending your computer derivative value and for now I placed their funny function just to remind you to fill that out.inimum distance that we will assume to be a valid measurement value. So any distances below 20.

## Finding cylinders in the scan data

And here's my solution. So all you have to do is you grab the left value, you grab the right value, and then if those values are both larger than the minimum distance. Then you compute the derivative, which is actually difference quotient, and you append this, and if either one of those two is smaller than the minimum distance, well, we don't care about the real derivative at that position, so we'll just return. Zero in that case, because that means later we will not detect the start or the end of a cylinder. So now let's run this and here's the result for scan #7. So you see we have those six spikes going down here. Each representing one cylinder in the real world, and our derivative produces for every falling edge and negative peak and for every rising edge the positive peak. And if you have a look at the range of those values we can see. That the pigs are stronger than 100 and minus 100, so we will just use a threshold of plus minus 100 to detect the falling or rising edge caused by a cylinder. Let's run this for another scan. So for scan #235 we have the situation that there is one of those peaks of erroneous measurement values. So in the measurement data there's here the signal for the cylinder and shortly before that. We have fun of those peaks going down to 15 or something like so. If you look at our derivative, we see that our detector worked perfectly, so it ignored this strong peak, but it responded to the falling edge caused by the cylinder. So the next thing we'll have to do. Was to write a program which detects those signals and determines the start and the end of every cylinder in the scan. But there's one caveat with that. Remember, there were situations in our skin. But it was one cylinder in the foreground. There was a small cylinder which was partially occluded in the background. So in that case our scanner hits the wall, then it starts to hit this background object, then the foreground object. And then it's going back to the wall again. So in terms of measurement data, we'll have this background measurement, then we will have a negative slope for the first cylinder and then a second negative slope for the second cylinder in the foreground. And then it will go back up again. So in this case our method will indicate us there's a left start when there is again a left start and then the right the end of the cylinder, meaning we have to cope with situations which are. Unusual. So instead of obtaining left, right, left, right, left, right and so on, we might also get left, left, right. Or we also might get left, right. So here's my strategy to solve that problem. Say our original signal is like that. Do you have a cylinder in the background, then another cylinder and then it goes back up again? So are derivative will be like that the peak here, now the peak here and a positive peak here and our threshold? Will indicate a left, a left, and a right edge. Now these points are all discreet, so I might have those five points here and two points here, four points here. So what I'm interested in is this point. So it is the average depth. And the average rate that will indicate where the cylinder is. So my strategy is as follows. I have one variable called on cylinder where a memorized if I'm currently on a cylinder and as long as I'm on a cylinder. Will add up the race and the depths. The Heffer rate counter. I have a sum of the race and have a sum of the debt. So when I start I'm not on a slender so this will be false. I just walk through those values. You say this is value 0 and add value. Wife, my derivatives indicates a def jam, so I'll switch to on cylinder mode true. Now initialize the race so zero so far, the sum of race zero and the sum of the death. So now for this value. Five, say the depth would be 50, so that gives up. I have now wondering, that's right #5 and some of the deaths will be 50. I go over to #6. As long as it is true, I will add up those builders. Have left two race summer phrase will be 5.6 and the sum of death will be 50 + 50. Now when I have another depth jump I will just discard all the measurements I made so far so the one cylinder stays true. Your chest reset, everything start over working and now I have to .789 and 10 and it will go on like that so it will sum up the race. One ray say the death here would be 40. Some of the rays will be seven depth 40 and it will go on like that, and when I'm here then I will have counted 4 rays. The sun will be 7 + 8 + 9 + 10, so it will have four death values of exactly 40. In this example, that's my derivative indicates a positive edge. Then we'll just compute the average rate, which will be 7 + 8 + 9 + 10 / 4, and that's 8.5. And the average depth which will be 40. And I will store this as a cylinder indicating average rate and average depth. And there's one caveat with that. Remember, when you sum up the race, you might also have those erroneous measurements. Here you have to make sure that you don't incorporate these meaning in that case. They don't count up the race, don't add up the sum of race and don't add this value to the depth. So in the end when it jumps back again here I set the on cylinder to false and as long as it is false I do not add up anything and I'll start over again. When I detect the next falling edge. So I want you to program this and so you'll find the file find selling just question. And in this file I prepared everything already for you, so you have to compute derivative function which is just the function that being computed previously. You have to find cylinders, which you'll have to implement, and down here there's the main function, and the main function is almost identical to the previous main function. In addition to the previous file, we now also have the depth jump variable. Which indicates the threshold we are using for finding a deck jump edge in our scan data. And that's in the last time we do have this minimum value distance I load the log file I picked the scan. As in the previous data, I compute the derivative, I compute the cylinders. This is what you have to implement. Then I've locked the scan and also the cylinders, and let's have a look at the function you'll have to implement. Here's 5 cylinders. It gets the scan. It gets derivative and thresholds for detecting the jump and the invalid range measurement values and should produce a cylinder list. I start by assigning one cylinder the value of faults and the sum of raise the sum of that. And the race, which are initialized to zero, and then for every point in the scan derivative. And this is where you'll have to implement the strategy, which is we're looking at. But you'll have to remove this just for fun. I'm generating some. The leaders in this list, which then show up in the graph that this program produces. When you run it, you'll see that result. So this is the original scan, and the red dots are the detected Skylanders. These are not the correct results. So after you implemented the correct version you will get a result like that. So if you have original scan data and here the points which indicate the cylinders and their exact locations.

# Computing cartesian coordinates.

There's one more thing I'll ask you to do, but fortunately that is really really so we now do have this robot and the robots coordinate system is like this. And as I told you earlier, the beams count like that and back here is an. Area where no measurements are. Now look at chest. Computed is the rate indices that point to the cylinders. And the rich and this here starts at index zero and it ends at index 659. So this may be index 400. I want you now to produce the XY coordinates in the robots. Coordinate system. So for every detected cylinder, given array index and range, I want you to produce the X&Y in the robots coordinate system. And for converting the rain index to an angle, there's a function provided. In the legal field which is called legal log file dot beam index to angle and this will convert the index like 400 here to an angle measured in red. There's one modification that you have to apply to the range measurement. Remember the robot measures some cylinder, so these are the rates and from that you would expect the range measurement which looks like that. This should be this rain. And then these other race, they are closing, so it should look somehow like this. But as you probably noticed in reality the outcome of our lighter something like that, even with the P. Being here, I didn't know our algorithm. We just computed the average of all those values, which then is here. So in that case we may get something like that. As it turns out, this is the correct. For the cylinders distance, so the cylinder is here and you would have to apply offset of the cylinder radius which is 55 millimeters but in addition they have fly a correction. Here, empirically, I determined that this correction that we have to apply maybe something like 19 millimeters and so whenever you convert one of your resulting branches. For XY coordinates, remember that you get something like this range and before converting it to XY you will have to add up 90 millimeters and this is called the cylinder offset. So in this is the final modification. I'll ask you for the program is fine. Cylinders Cartesian and it consists of the compute derivative which you know already defined, cylinders which you'll just have to fill in from your previous solution and this new function. Which you'll have to program. So this function gets a list of all the cylinders that were detected in one scan. This might be zero or more cylinders, and then for every cylinder takes the top of beam index range. And from there has to compute an XY coordinate. And just for now I have put 00 here. Now let's have a look at the main function which I modified a little bit so there's two minimum bellot distance and the depth jump. There there's not the cylinder also, which is a global variable you may use in here. In the fact that you have, the program still opens the log file and reads the log file, but now instead of drawing the response, it will produce a file called cylinders. And it feels right for every single scan one line to this file which starts with DC, meaning detected cylinders, and will then list all the Cartesian coordinates. Cylinders you've just about. So in the loop he calls the computation of the derivative, he calls the fine cylinders, and then it calls your new function compute Cartesian coordinates.