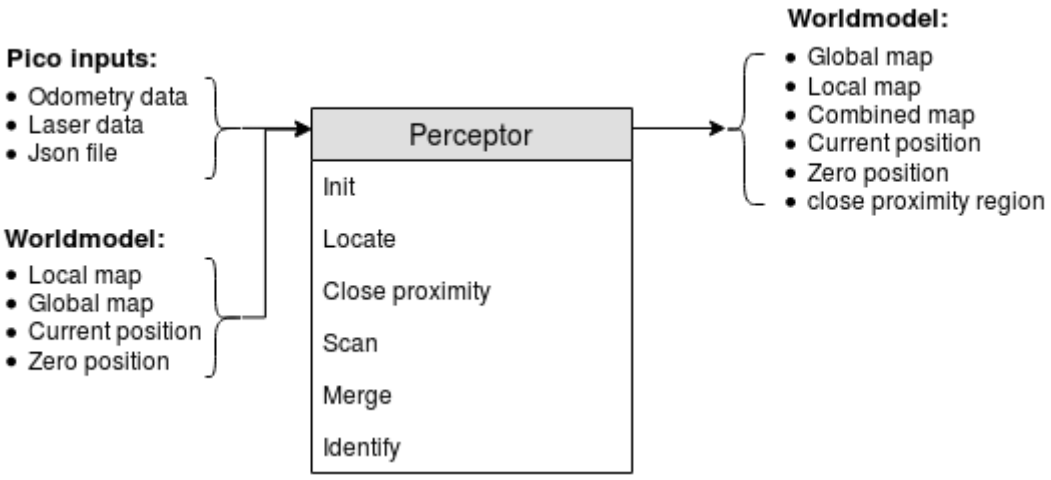
Perceptor functionality

The perceptor receives all of the incoming data from the pico robot and converts the data to useful data for the worldmodel. The incoming data exists of odometry data obtained by the wheel encoders of the pico robot. The laserdata obtained by the laser scanners. A Json file containing the global map and location of the cabinets, this file is provided a week before the hospital challenge. Moreover, the output of the perceptor to the world model consists of the global map, a local map, a combined map, the current/zero position and a close proximity region. The incoming data is handled within the perceptor by the following functions.



Init

The pico robot saves the absolute driven distance since its startup. Therefore, when the software starts it needs to reinitialize the current position of the robot. If the robot receives its first odometry dataset it saves this position and sets it as zero position for the worldmodel. This happens only once when the software is started.

Locate

This function makes use of the zero-frame which is determined by the init function. Once the odometry data is read a transformation is used to determine the position of the robot with respect to the initialized position. This current position is outputted to the worldmodel.

Close proximity

Dynamic objects are not measured by the local map. To prevent collisions with the robot and a dynamic objects or walls, a close proximity region is determined. This region is described as a circle with the configured radius around the robot. The function returns a vector of booleans to the worldmodel if the robot is to close to an object. To make this function more robust the laser data which lies inside of the robot is excluded.

Scan

The laserdata is read from the sensor, however, this data is in polar coordinates. Therefore, the data is first transformed to Cartesian coordinates. Next, the data is resampled so there is a minimum set distance between all of the data points. This resampled data is used to calculate the angles between all the consecutive points. To determine which data points represent a wall the data is split in different clusters. The data is splitted using an average angle of the cluster and the angle of the next point. Therefore, the data is split at the corners of each wall. Lastly, the points of each cluster are

marked as a wall with floating points at both sides of the walls also the position of these floating points is stored as well. This stored information of the end points of walls is defined as current map.

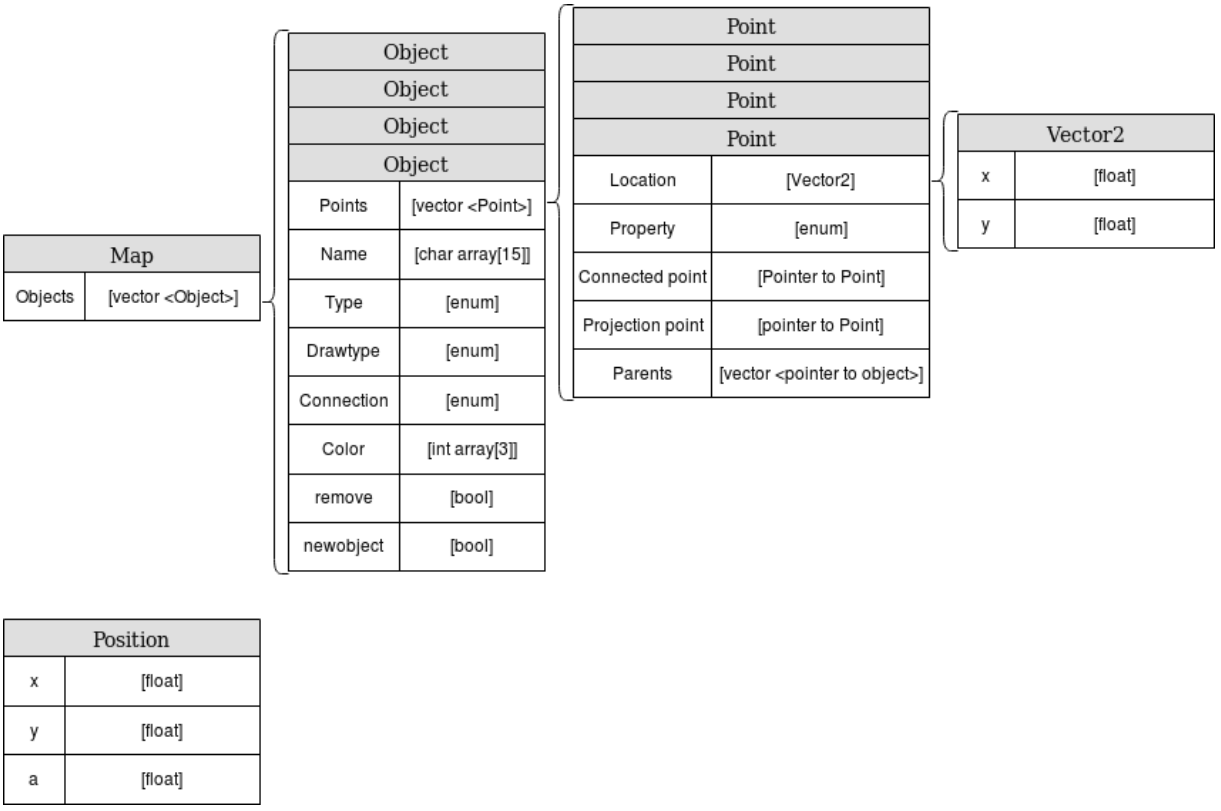
Merge

In this function the new laser data is merged with an existing map to form a more robust and complete map of the environment. Therefore, laser data in the form of a current map created by the scan function is imported. Furthermore, the previous created output of the merge function is imported as well, which is called the local map. Firstly, the previous created map is transformed to the current position of the robot. Secondly, similar walls are merged. Walls are considered to be similar if they are parallel to each other, have a small difference in angle or are split into two pieces. If two walls are close to each other but one has a smaller length they are merged as well. The merge settings are stored in the configuration file. The different merge cases can be seen in figure ... . Once similar walls are merged, the endpoints of walls are connected to form the corners points of the room. Each point of a wall has a given radius, and if another point has a distance to this point which is smaller than its radius then the points will be connected. To improve the robustness of the local map the location of these corner points is mainly based on the location of the corner points from the previous local map, which ensures rejection of measurement errors in the laser data. Furthermore, the wall points that are not merged or connected at the end of the function will be removed. Therefore, the local map will only consist of walls which are connected to each other.

Identify

The functionality of this function is to identify the property of the points in the local map. For instance corner points can be convex or concave. This property is later used to help identify doors, objects or cabinets in the local map. The position of the robot determines if the corner point is convex or concave. With this property information the map is scanned for doors. For the escape room challenge a door is identified as two convex points close to each other. A door is defined between two walls, these walls should be approximately in one line. Also, the corner points cannot be from the same wall to further increase the robustness of the map. It is unlikely that the local map immediately contains two convex points which can form a door. Therefore, a possible door is defined, so the robot can drive to the location and check if there is a real door at this position. There are multiple scenarios where a possible door can be formed. Such as, one convex point and one loose end, two loose ends or a loose end facing a wall. Concave points can never form a door and are therefore excluded. When forming a possible door the length of the door and the orientation of the walls is important as well.

Data structures used by the perceptor



Map struct		
Variable	**Type**	**Usage**
Objects	[Vector <Object>]	To define the map. The map exists of multiple objects which can be a different type.
Function	**I/O**	**Description**
Transform	[Map| x| y| a] > [Map]	Transform all the object in the map to a new location and orientation. First rotate then translate.
Print	[Map] > [-]	Print the names of all the objects in the console can be used for debugging.
removeobjects	[Map] > [Map]	Reorder the vector of objects so all the objects which have the remove flag set are at the end of the vector an then remove these objects from the vector.
setobjectsold	[Map] > [Map]	This function sets all the newobject variables to false which is used to distinguish between the data from the objects from the new scan and

		the objects already in the local map.

Object struct		
Variable	**Type**	**Usage**
Name	[char array [15]]	Visualization: Identification
Type	[enum]	Identification used to separate different objects [wall | door | test | origin | robot | dynamicobstacle | staticobstacle |safeDis]
Points	[vector <Point>]	To define objects which consist of multiple points. Walls and doors consist of 2, other objects such as the origin, the robot, arrows and so on can consist of more points.
Drawtype	[enum]	Visualization: define if the points have to be connected with lines [points | lines]
connection	[enum]	Visualization: define if the first and last point have to be connected with a line [open | closed]
Color	[int array [3]]	Visualisation: specify the color of the object
remove	[bool]	Map remove function: determine if the object will be removed when the next removeobjects is called.
newobject	[bool]	Do distinguish between the objects that are already in the localmap and the objects from the new scan.
Function	**I/O**	**Description**
angle	[Object] > [float]	Calculate the angle of an object relative to 0. Works only when the object has 2 points eg wall or door.
length	[Object] > [float]	Calculate the length of an object. Works only when the object has 2 points eg wall or door.
smallestrelativeangle	[Object| Object] > [float]	Calculate the smallest angle between 2 objects which both have 2 points.
averageperpendiculardistance	[Object| Object] > [float]	Calculate the average perpendicular distance between 2 objects which both have 2 points.

anglebetween	[Object| Object] > [float]	Calculate the angle between 2 objects measured from 1 direction.
gapdistance	[Object| Object] > [float]	Calculate the gap distance between 2 objects which both have 2 points. This is used when 2 objects are approximately parallel and have a small average perpendicular distance.
transform	[Object| x| y| a] > [Object]	This rotates and then translates all the points in this object with angle a an offset x and y.

Point struct		
Variable	**Type**	**Usage**
Location	[Vector2]	To describe the location of the point in a XY frame
Property	[enum]	To describe the property of a point [floating | convex | concave | connected]
Connected point	[pointer to Point]	To specify to which point it is connected
Projection point	[pointer to Point]	A reference to the point it is connected to or essentially on top of.
Parents	[pointer to Objects]	References to all objects which have a point at this location
Weight	[Float]	A weight to describe how certain a point is in the correct location
Function	**I/O**	**Description**
sameparentobject	[Point| Point] > [bool]	To check if two points have the same parent object.

Vector2 struct		
Variable	**Type**	**Usage**
x	[Float]	To store the x component of the vector.
y	[Float]	To store the y component of the vector.
Function	**I/O**	**Description**
Operators | + | - |	[Vector2| Vector2] > [Vector2]	To add or subtract 2 vectors.
Operators | / | * |	[Vector2| float] > [Vector2]	To multiply or divide a vector with a number.
Operator | == |	[Vector2| Vector2] > [bool]	Determine if 2 vectors are the same.

Distance	[Vector2] > [float]	Calculate the Distance between 2 vectors.
Length	[Vector2] > [float]	Calculate the length of the input vector.
Angle	[Vector2] > [float]	Calculate the angle of the input vector relative to 0.
dot	[Vector2|Vector2] > [float]	Calculate the dot product of 2 vectors.
unit	[Vector2] > [Vector2]	Calculate the unit vector of a given vector.
transform	[Vector2| x| y| a] > [Vector2]	Rotate the input vector with angle a and then translate this vector with x and y.

Position struct		
Variable	**Type**	**Usage**
x	[float]	To store the x component of the position.
y	[float]	To store the y component of the position.
a	[float]	To store the angle component to the position.
Function	**I/O**	**Description**
Operators | + | - |	[Position] > [Position]	To do simple calculations with a position variable eg the destination or current position.