API

Navigation Module

Module present on both robots, with modifications.

Void SetInitialPosition()

Void SweepArea

This function will sweep the field in a pre-defined pattern. It will succesively call Navigate on those points.

Using various sensors, this function will attempt to decide what the initial position is, and position the robot correctly.

Navigate (float x, float y, bool ignoreBalls, bool isBowlLocation, long timeout)

Given a starting position (A) (read from odometry) and an end position (B), it will navigate from A to B while avoiding obstacles. It will accept a parameter specifying whether to ignore balls or not (if supported by the robot), and whether the destination is the bowl – in which case it will stop at a distance from the bowl. The third parameter is the timeout – after this much time, we don't want to navigate and think are stuck.

Return value: different error/success codes depending on how the operation finished.

RecalibratePosition (depends on the actual strategy, this is low priority)

This function will be called in an emergency – it will try to reposition the robot if we detect that odometry is not valid anymore)

Void CorrectError()

This function will adjust the odometry each time we receive reliable information about our real position, either from the lines on the floor, compass (different values in different squares) or the wall.

Float FindPass(int Direction) //Left or right

While pointing at an obstacle (non-ball), this function will find the first opening, in the indicated direction, that the robot can pass through. Ideally this function will be called twice, and find the nearest one.

It returns the angle at which the robot can safely move at, without bumping into anything.

Load/Transfer module

This module, only present on the Loader, contains functionality required to load balls into the bay. It also contains the module that is responsible for transferring the ball to the Scooper.

Float AcquireBallHeading ()

Starting in a position where the sensor is pointing at a ball, it will find the ball edges and then return the heading of the ball.

Returns the correct ball heading.

Int LoadBall(float distance, float heading)

Given the distance to the ball and the correct heading, it will attempt to load the ball in the bay.

It returns the ball color in case of success, error in case the ball is not loaded.

Int SummonScooper(long timeout)

Communicate with the Scooper and orchestrate correctly such that the Scooper will be positioned correctly to start transferring the ball. Allow at most timeout for the Scooper to arrive (set to 0 for infinity)

Returns success if Scooper is at the right position and an Error code if something went horribly wrong.

Int TransferBall()

Synchronizes with the Scooper in order to transfer the ball from the bay to the Scoop. It includes the (safe) lifting of the Scoop.

Returns 1 if the ball is transferred correctly. 0 if there is a problem.

Int NeutralizeScooper(long timeout)

This function will coordinate with the Scooper and make sure that it is not in the way, and that its US sensor is not interfering with the Loader's. It returns Success when Scooper is "neutralized", Error if something went wrong.

Fault Tolerance and Recovery Module

This module contains function that detect and deal with exceptional situations. They are called from one task, regularly.

CheckMovingStatus – Check if robot is stuck

Check Scoop/Arm/Loading Arm status – Check if the moving parts of the robot are in the correct positions

Check odometry – This function will try to assess the current position by other means than the x,y,theta values it has, and trigger a flag if something is detected to be wrong.

Control Module

Void RotateHelper(float radians, int power)

Rotate, without a feedback loop, the specified number of degrees.

Void Rotate(float radians, int power)

Rotate (in a feedback loop) **precisely** radians, at given power. This algorithm will oscilate a little bit, but converge to the desired value (usually within less than 10 oscillations) or stop after a certain number of attempts (40).

Void GoToHeading(float newHeading, int power)

Rotate (in a feedback loop) **precisely** to the specified heading, at given power. This algorithm will oscilate a little bit, but converge to the desired value (usually within less than 10 oscillations) or stop after a certain number of attempts (40).

Void GoToXY(float x, float y, int power)

Move (in a straight line) to the position indicated by x and y.

CorrectPosition – At certain intervals, the robot will know, either from odometry or from other functions that its position is not what it is supposed to be.

Odometry module

This module continuously updates the position and heading of the robot (in global variables x,y,theta (all floats)), based on the movement of the wheels.

The loader will have the same navigation code, with minor changes. The Scooper code is trivial, and all other actions it takes are controlled by the Loader.

Task distribution

1. Odometry – computes all code required to update the x, y and theta values. High priority
2. Drive – controls the high-level logic of the robot, including the state machine.
3. PlumbingTask – this task updates various values (non-odometry) and flags used by other routines, that the code uses – sensor readings, timers, etc. It will call functions that perform these tasks. It might be merged with Odometry, under the name of UpdateValues

Communication is NOT a separate task, and it is always used as functions called exclusively from the Drive task.