Robot Control Class

The robot control class is responsible for selecting models from all the information found by both line-finding nodes, and calculating an ideal velocity based in the finite state machine and a fuzzy controller. The logic is simple: first, receives a bunch of message coming from back and front nodes and initialize the algorithm calculating an average distance between rows and average slope. For this initialization, we suppose the robot will begin its movement approximately at the center of the two rows, and parallel to them, being able to filter the majority of models found. After this step, the algorithm will have the first 4 models (2 to the right and 2 to the left), then, using these first models, we can compare all models received with them. With the hypothesis that the robot will move in a slowly and continuous fashion, the models selected for the next iteration must be a tiny delta from the one previously. If at least one model was found using this method, we can calculate the other models using the distance information. Finally, we add all points in the field that are close to the model, this way we know the precise beginning and end (for calculated models, it receives the found models points, this way we assume they will begin and end aligned with the others)

Public Methods

◆ RobotControl( )

```
RobotControl::RobotControl(
                                const int NLines,
                                const int NTimesTurn,
                                const double MVel,
                                const double BSize
                              )
```

Default constructor that declares vectors and the finite state machine. Sets “Number of lines” to define the number of models to be selected (must be pair), “Number of times to turn” to stop the robot at the right time, “Max Vel” to control how fast the robot goes, and “Body size” to set the robot’s length.

◆ clearModels( )

```
void RobotControl::clearModels ( )
```

Clear all models attached

◆ frontLinesMessage( )

```
void RobotControl::frontLinesMessage ( const visualization_msgs::Marker & msg )
```

Receives message from front line finder node

Parameters

msg is the message coming from front node

◆ backLinesMessage( )

void RobotControl::backLinesMessage (const visualization_msgs::Marker & msg)

Receives message from back line finder node

Parameters

msg is the message coming from back node

◆ frontPointsMessage( )

void RobotControl::frontPointsMessage (const visualization_msgs::Marker & msg)

Receives the clustered points from find lines node

Parameters

msg is the message containing all the points

◆ backPointsMessage( )

void RobotControl::backPointsMessage (const visualization_msgs::Marker & msg)

Receives the clustered points from find lines node

Parameters

msg is the message containing all the points

◆ selectModels( )

std::pair<std::vector<Model>, std::vector<bool>> RobotControl::selectModels ( )

Return all 4 models found (or calculated) and a vector of bool to inform if the model in a given index was found (bool = false) or calculated (bool = true)

◆ getWheelsCommand( )

std::pair<std_msgs::Float64, std_msgs::Float64> RobotControl::getWheelsCommand( )

Uses the saved models and the robot's finite state machine to calculate the output

Private Methods

◆ addMsgModels( )

void RobotControl::addMsgModels (const std::vector<Model> & modelsInMsg)

Increments, fuse and add new models to the weighted models vector

Parameters

modelsInMsg is the models you want to add to the object

◆ translateAxis( ) [1/2]

```
visualization_msgs::Marker RobotControl::translateAxis
(
  const visualization_msgs::Marker & msg,
  const double newOX,
  const double newOY
) const
```

Translate points in ‘msg’ to new X and Y origin

Parameters

msg is message containing the points
newOX is the new X coordinate origin
newOY is the new Y coordinate origin

◆ translateAxis( ) [2/2]

```
std::vector<Model> RobotControl::translateAxis
(
  const std::vector<Model> & msg,
  const double newOX,
  const double newOY
) const
```

Translate points in ‘msg’ to new X and Y origin

Parameters

msg is message containing the points
newOX is the new X coordinate origin
newOY is the new Y coordinate origin

◆ rotateAxis( ) [1/2]

```
visualization_msgs::Marker RobotControl::rotateAxis
(
  const visualization_msgs::Marker & msg,
  const double angleRot
) const
```

Rotate points in ‘msg’ in the Z-axis by the angle ‘angleRot’

Parameters

msg is message containing the points
angleRot is the angle amount to rotate

◆ rotateAxis( ) [2/2]

```
std::vector<Model> RobotControl::rotateAxis
(
  const std::vector<Model> & msg,
  const double angleRot
) const
```

Rotate points in ‘msg’ in the Z-axis by the angle ‘angleRot’

Parameters

msg is message containing the points
angleRot is the angle amount to rotate

◆ getFoundLines( )

```
std::vector<Model> RobotControl::getFoundLines
                                   ( const visualization_msgs::Marker & msg ) const
```

Extract models from message points

Parameters

msg is message containing the points

◆ friend operator << ( )

```
std::ostream & operator << (std::ostream & out, const RobotControl & rc )
```

Print robot control object

Parameters

out is where to print, normally terminal
rc is the object to be printed