Autonomous Rescue Robot

Michael Balas

January 7, 2019

The purpose of this work is to design and implement a portion of the software for an autonomous rescue robot. This document shows the complete specification, which is the basis for implementation and testing. The inspiration for the project is real life rescue robots, which are used when a disaster occurs and the conditions are too dangerous for human rescuers.

Constants Module

Module

Constants

Uses

N/A

Syntax

Exported Constants

MAX_X = 180 *//dimension in the x-direction of the problem area*
MAX_Y = 160 *//dimension in the y-direction of the problem area*
TOLERANCE = 5 *//space allowance around obstacles*
VELOCITY_LINEAR = 15 *//speed of the robot when driving straight*
VELOCITY_ANGULAR = 30 *//speed of the robot when turing*

Exported Access Programs

none

Semantics

State Variables

none

State Invariant

none

Point ADT Module

Template Module

PointT

Uses

Constants

Syntax

Exported Types

PointT = ?

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
PointT	real, real	PointT	InvalidPointException
xcrd		real	
ycrd		real	
dist	PointT	real	

Semantics

State Variables

xc: real
yc: real

State Invariant

none

Assumptions

The constructor PointT is called for each abstract object before any other access routine is called for that object. The constructor cannot be called on an existing object.

Access Routine Semantics

PointT(x, y):

- transition: $xc, yc := x, y$
- output: $out := self$
- exception $exc := ((\neg(0 \leq x \leq \text{Contants.MAX\_X}) \vee \neg(0 \leq y \leq \text{Constants.MAX\_Y})) \Rightarrow$ InvalidPointException)

xcrd():

- output: $out := xc$
- exception: none

ycrd():

- output: $out := yc$
- exception: none

dist(p):

- output: $out := \sqrt{(self.xc - p.xc)^2 + (self.yc - p.yc)^2}$
- exception: none

Region Module

Template Module

RegionT

Uses

PointT

Syntax

Exported Types

RegionT = ?

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
RegionT	PointT, real, real	RegionT	InvalidRegionException
pointInRegion	PointT	boolean	

Semantics

State Variables

lower_left: PointT *//coordinates of the lower left corner of the region*
width: real *//width of the rectangular region*
height: real *//height of the rectangular region*

State Invariant

None

Assumptions

The RegionT constructor is called for each abstract object before any other access routine is called for that object. The constructor can only be called once.

Access Routine Semantics

RegionT(p, w, h):

- transition: $lower\_left, width, height := p, w, h$
- output: $out := self$
- exception:

$$\begin{aligned}
exc := \neg(&w > 0 \wedge \\
&h > 0 \wedge \\
&(p.\text{xcrd}() + w) \leq \text{Constants.MAX_X} \wedge \\
&(p.\text{ycrd}() + h) \leq \text{Constants.MAX_Y}) \Rightarrow \text{InvalidRegionException}
\end{aligned}$$

pointInRegion(p):

- output: $out := \exists(q : \text{PointT} | q \in \text{Region} : p.\text{dist}(q) \leq \text{Constants.TOLERANCE})$
- exception: none

Local Functions

Region: set of PointT

$$\begin{aligned}
\text{Region} \equiv \cup(&q : \text{PointT} | \\
&lower_left.\text{xcrd} \leq q.\text{xcrd} \leq (lower_left.\text{xcrd} + width) \wedge \\
&lower_left.\text{ycrd} \leq q.\text{ycrd} \leq (lower_left.\text{ycrd} + height) : \{q\})
\end{aligned}$$

Generic List Module

Generic Template Module

GenericList(T)

Uses

N/A

Syntax

Exported Types

GenericList(T) = ?

Exported Constants

MAX_SIZE = 100

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
GenericList		GenericList	
add	integer, T		FullSequenceException, InvalidPositionException
del	integer		InvalidPositionException
setval	integer, T		InvalidPositionException
getval	integer	T	InvalidPositionException
size		integer	

Semantics

State Variables

s: sequence of T

State Invariant

$|s| \leq$ MAX_SIZE

Assumptions

The GenericList() constructor is called for each abstract object before any other access routine is called for that object. The constructor can only be called once.

Access Routine Semantics

GenericList():

- transition: $self.s :=<>$
- output: $out := self$
- exception: none

add(i, p):

- transition: $s := s[0..i-1] || < p > || s[i..|s|-1]$
- exception: $exc := (|s| = \text{MAX\_SIZE} \Rightarrow \text{FullSequenceException} \mid i \notin [0..|s|] \Rightarrow \text{InvalidPositionException})$

del(i):

- transition: $s := s[0..i-1] || s[i+1..|s|-1]$
- exception: $exc := (i \notin [0..|s|-1] \Rightarrow \text{InvalidPositionException})$

setval(i, p):

- transition: $s[i] := p$
- exception: $exc := (i \notin [0..|s|-1] \Rightarrow \text{InvalidPositionException})$

getval(i):

- output: $out := s[i]$
- exception: $exc := (i \notin [0..|s|-1] \Rightarrow \text{InvalidPositionException})$

size():

- output: $out := |s|$
- exception: none

Path Module

Template Module

PathT is GenericList(PointT)

Obstacles Module

Template Module

Obstacles is GenericList(RegionT)

Destinations Module

Template Module

Destinations is GenericList(RegionT)

SafeZone Module

Template Module

SafeZone extends GenericList(RegionT)

Exported Constants

MAX_SIZE = 1

Map Module

Module

Map

Uses

Obstacles, Destinations, SafeZone

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
init	Obstacles, Destinations, SafeZone		
get_obstacles		Obstacles	
get_destinations		Destinations	
get_safeZone		SafeZone	

Semantics

State Variables

$\mathit{obstacles}$: Obstacles
$\mathit{destinations}$: Destinations
$\mathit{safeZone}$: SafeZone

State Invariant

none

Assumptions

The access routine init() is called for the abstract object before any other access routine is called. If the map is changed, init() can be called again to change the map.

Access Routine Semantics

init(o, d, sz):

- transition: $\mathit{obstacles}, \mathit{destinations}, \mathit{safeZone} := o, d, sz$

- exception: none

get_obstacles():

- output: $out := obstacles$
- exception: none

get_destinations():

- output: $out := destinations$
- exception: none

get_safeZone():

- output: $out := safeZone$
- exception: none

Path Calculation Module

Module

PathCalculation

Uses

Constants, PointT, RegionT, PathT, Obstacles, Destinations, SafeZone, Map

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
is_validSegment	PointT, PointT	boolean	
is_validPath	PathT	boolean	
is_shortestPath	PathT	boolean	
totalDistance	PathT	real	
totalTurns	PathT	integer	
estimatedTime	PathT	real	

Semantics

is_validSegment(p_1, p_2):

- output: $out := \forall(i : \mathbb{N} | 0 \leq i < \text{Map.get\_obstacles.size()} : \text{is\_valid\_segment\_for\_region}(p_1, p_2, i))$
- exception: none

is_validPath(p):

- output:

$$
\begin{aligned}
&out := \\
&\quad \text{Map.get\_safeZone.getval(0).pointInRegion}(p.\text{getval}(0)) \wedge \\
&\quad \text{Map.get\_safeZone.getval(0).pointInRegion}(p.\text{getval}(p.\text{size}() - 1)) \wedge \\
&\quad \forall(i : \mathbb{N} | 0 \leq i < \text{Map.get\_destinations.size()} : \text{pathPassesThroughDestination}(p, i)) \wedge \\
&\quad \forall(i : \mathbb{N} | 0 \leq i < p.\text{size}() - 1 : \text{is\_validSegment}(p.\text{getval}(i), p.\text{getval}(i + 1)))
\end{aligned}
$$

- exception: none

is_shortestPath(p):

- output:

$$out := \forall(q : \text{PathT} | \text{is\_validPath}(q) : \text{is\_validPath}(p) \wedge \text{totalDistance}(p) \leq \text{totalDistance}(q))$$

- exception: none

totalDistance(p):

- output:

$$out := +(i : \mathbb{N} | 0 \leq i < (p.\text{size}() - 1) : p.\text{getval}(i).\text{dist}(p.\text{getval}(i + 1)))$$

- exception: none

totalTurns(p):

- output:

$$out := +(i : \mathbb{N} | 0 \leq i < (p.\text{size}() - 2) : \text{angle}(p.\text{getval}(i), p.\text{getval}(i+1), p.\text{getval}(i+2)) \neq 0 : 1)$$

- exception: none

estimatedTime(p):

- output: $out := \text{linear\_time}(p) + \text{angular\_time}(p)$
- exception: none

Local Functions

is_valid_segment_for_region: PointT × PointT × integer → boolean
is_valid_segment_for_region(p_1, p_2, i) ≡

$$\forall(t : \mathbb{R} | 0 \leq t \leq 1 : \neg \text{Map.get\_obstacles.getval}(i).\text{pointInRegion}(tp_1 + (1 - t)p_2))$$

pathPassesThroughDestination: PathT × integer → boolean
pathPassesThroughDestination(p, i) ≡

$$\exists(q : \text{PointT} | q \in \text{Path} : \text{Map.get\_destinations.getval}(i).\text{pointInRegion}(q))$$

where Path $\equiv p$. This solution assumes that the sequence of points in the path include points within the destination regions. This assumption is fine, but if one decided not to make this assumption, then the definition of Path is a little more involved, as follows:

$$Path \equiv \cup(i : \mathbb{N} | 0 \leq i < (p.\text{size}() - 1) : \text{LineSeg}(p.\text{getval}(i), p.\text{getval}(i+1)))$$

where LinSeg: set of PointT, $\text{LinSeg}(p_1, p_2) \equiv \cup(t : \mathbb{R} | 0 \leq t \leq 1 : \{tp_1 + (1-t)p_2\})$.

angle: PointT $\times$ PointT $\times$ PointT $\rightarrow$ real
$\text{angle}(p_1, p_2, p_3) \equiv \cos^{-1}\left(\frac{\mathbf{u} \cdot \mathbf{v}}{||\mathbf{u}||||\mathbf{v}||}\right)$ where $\mathbf{u} = \mathbf{p_2} - \mathbf{p_1}$, $\mathbf{v} = \mathbf{p_3} - \mathbf{p_2}$, $||\mathbf{u}|| = p_1.\text{dist}(p_2)$ and $||\mathbf{v}|| = p_2.\text{dist}(p_3)$, with $\mathbf{p_i}$ being the vector from the origin to the point p_i for $i \in [1..3]$.

linear_time: PathT $\rightarrow$ real

$$\text{linear\_time}(p) \equiv +\left(i : \mathbb{N} | 0 \leq i < p.\text{size}() - 1 : \frac{p.\text{getval}(i).\text{dist}(p.\text{getval}(i+1))}{\text{Constants.VELOCITY\_LINEAR}}\right)$$

angular_time: PathT $\rightarrow$ real

$$\text{angular\_time}(p) \equiv +\left(i : \mathbb{N} | 0 \leq i < p.\text{size}() - 2 : \frac{\text{angle}(p.\text{getval}(i), p.\text{getval}(i+1), p.\text{getval}(i+2))}{\text{Constants.VELOCITY\_ANGULAR}}\right)$$