



Spacial Debris Collector Robot

An AI-based and autonomous debris collector satellite simulation



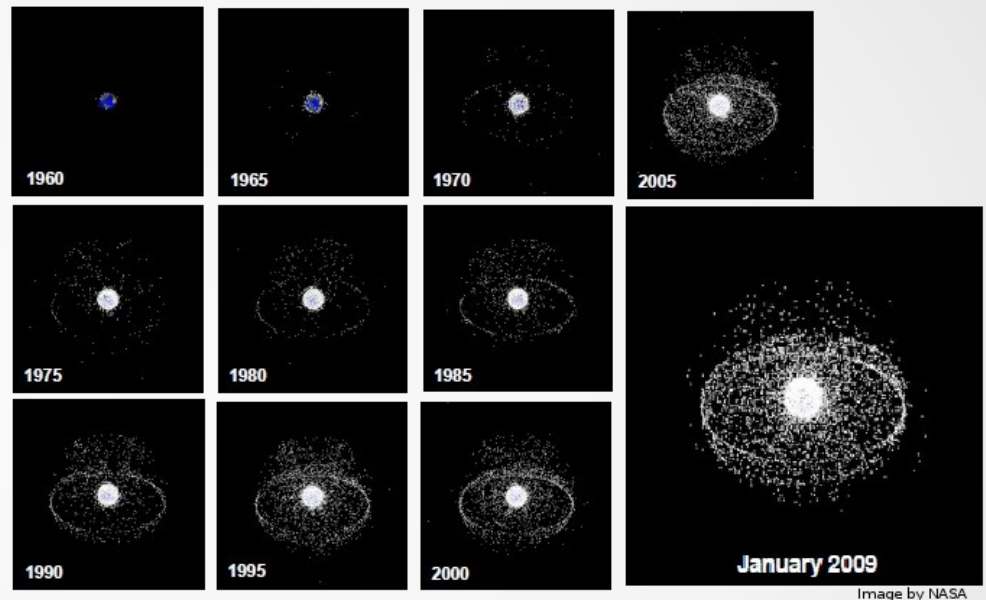
Treball Final de Grau

Author: David Fernández López

Tutor: Dr David Isern Alarcón

Space Debris

- Formed by thousands of different objects.
- Constant growth with no mitigation.
- Danger for both manned and unmanned crafts.
- Possible Kessler syndrome.



Proposed solution

- Autonomous Spacecraft.
- Transport out-of-order satellites to a safe area.

World Model

Position and Orbit

- Each object position is represented, in spherical coordinates, as the (r, θ, φ) triad.
- Orbits are represented as the (v, α, β) triad.

Object classification

- Objects are classified as Targets or Obstacles.
- Each Object is represented as a point (using spherical coordinates) and an orbit.

Robot representation

- The robot is represented as a point with an orientation, expressed as a Cartesian vector.

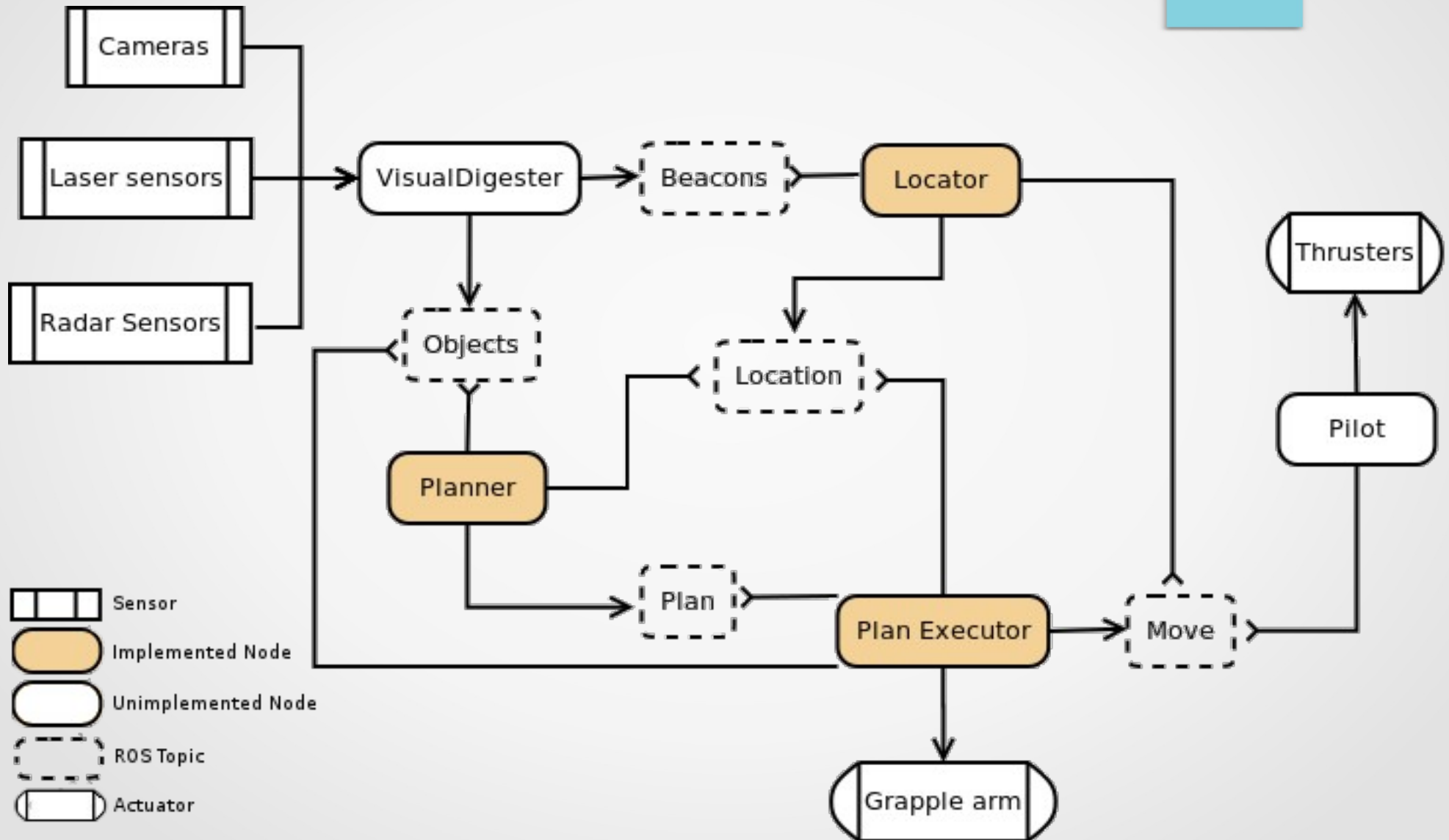
Robot Operating System (ROS)

- Open-source project.
- Implemented in C++.
- BSD License



- ROS provides libraries and tools to help software developers create robot applications.
- Uses a graph-like underlying structure.
- Provides easy communication between nodes through Topics and Services.
- Implements message queues, node tracking, support file generation and compiling tools.
- Programming language is C++.

Debris Collector Design



Planner node

- The planner node is the agent in charge of devising a plan to pick up the targets with the minimum fuel consumption possible.
- The planner node uses a genetic algorithm specifically tailored to its needs.
- The genetic algorithm is only run on demand.
- This node receives the position of nearby targets from other nodes, and sends the devised plan to the nodes that require it.

Pick up time calculations

Brute force algorithm (BFA)

Pick up times are randomly generated.

Pros: Much faster and shorter plans.

Cons: Higher result dispersion and higher costs.

Minimum distance algorithm (MDA)

Pick up times are chosen to minimize distance between the actual point and the Target.

Pros: Lower costs, more cohesive results and takes less iterations.

Cons: Extremely slow.

The mixed algorithm

- Initial population pick up times are calculated as in the MDA.
- During execution new pick up times are calculated as the time interval used in the parent.
- **The mixed algorithm completion times are similar to those of the BFA, with results close to those generated by the MDA.**

Genetic algorithm

1. Create Initial Population

3600	8000	11600	15000
0	1	2	3

3700	8500	12000	15600
1	3	2	0

2. Calculate costs

1	4	2	2	9
3600	8000	11600	15000	
0	1	2	3	

1	3	2	1	7
3700	8500	12000	15600	
1	3	2	0	

3. Check for convergence

Iterations > MAX ITERATIONS

$$\frac{|best_{i-1} - best_i|}{\max(best_{i-1}, best_i)} < DIFF$$

1	4	2	2	9
3600	8000	11600	15000	
0	1	2	3	
1	3	2	1	7
3700	8500	12000	15600	
1	3	2	0	
3700	7300			
1	2			

4. Create new population

3700	7300	10900	14900
1	2	0	3
3700	7300	10900	14900
1	3	0	2

5. Standard mutation

3700	7300	10900	14900
1	2	0	3
3700	7300	10900	14500
1	2	0	3

6. Time mutation

Locator node

- The locator node is the agent responsible for identifying the current position of the robot.
- The locator node uses a slightly modified particle filter algorithm.
- The particle filter algorithm implemented in this node runs permanently.
- This node receives the “locator beacons” distance to the robot from other nodes and sends the calculated position to the nodes that require it.

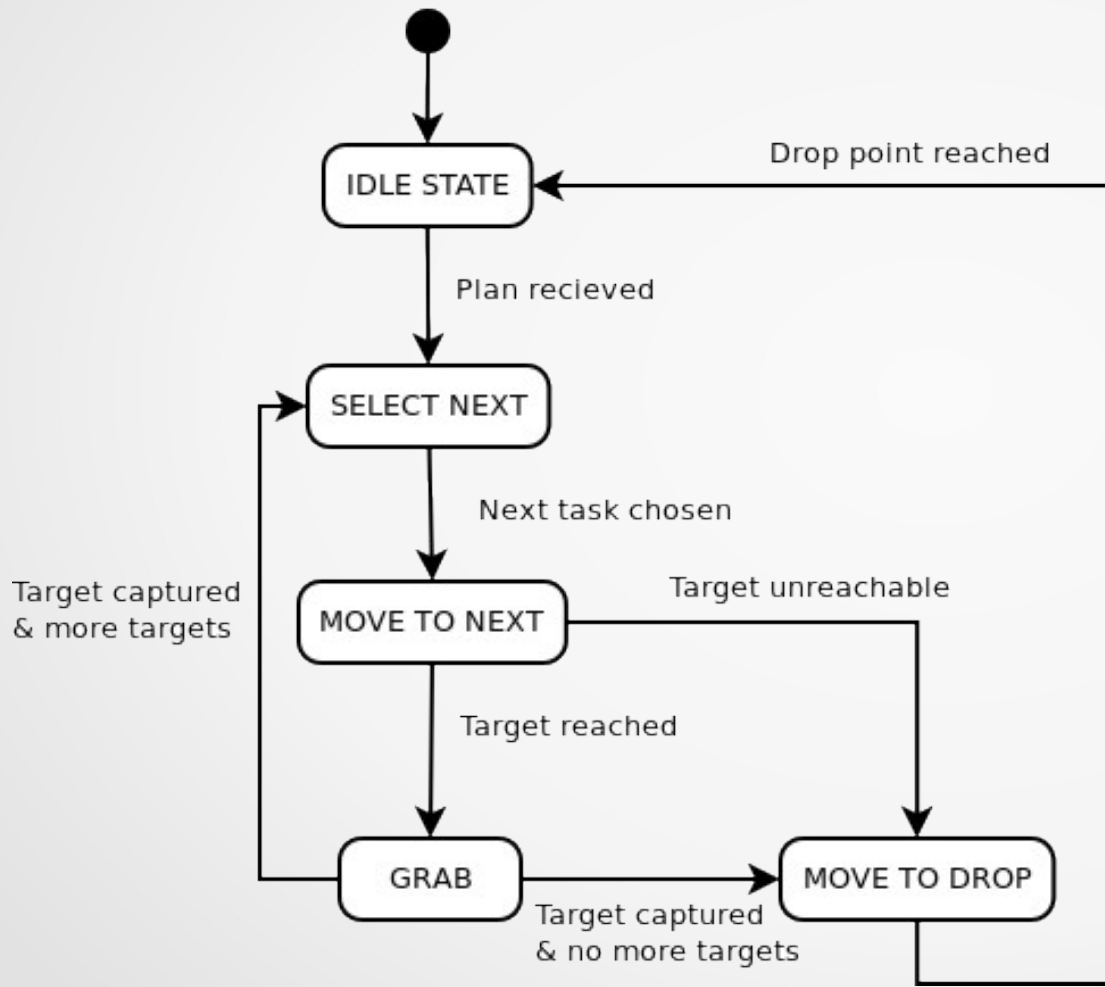
Simplified Scattered Particle Filter

- 1 - Create a number of particles with random position and orientation.
- 2 - Calculate the actual distance between each particle and the localizer beacons.
- 3 - Assign a weight to each particle according to the error between the actual distance and the expected distance to the beacons.
- 4 - Generate a new particle population by: resampling (with replacement) the previous population and applying scattering.
- 5 - Apply the same movement to the robot and to each particle.
- 6 – Return to 2.

Executer node

- The executer node is the agent responsible for carrying out the plan devised by the planner node.
- The executer node uses a finite state machine to implement its behavior.
- The executer node is also responsible for collision avoidance during flight.
- The executer node receives the plan and the location of nearby objects from other nodes and sends the required rotation and speed to apply to the nodes that require it.

Executor node behavior



- **SELECT NEXT** state selects new task and calculates target position at pick up time.
- The **GRAB** state keeps trying to grab the target until it succeeds.
- Unending finite state machine.

Perpendicular Elastic Force

- The Perpendicular elastic force uses a virtual repulsion force to avoid obstacles.
- When an obstacle comes close to the robot the virtual repulsion force is applied to steer the robot away.

Formulas

→ Movement vector

$$\vec{M} = \vec{V}_{robot} + \sum_{objects} \begin{cases} \vec{F}_r & \text{if } |\vec{D}| < Distance_s \\ \vec{0} & \text{otherwise} \end{cases}$$

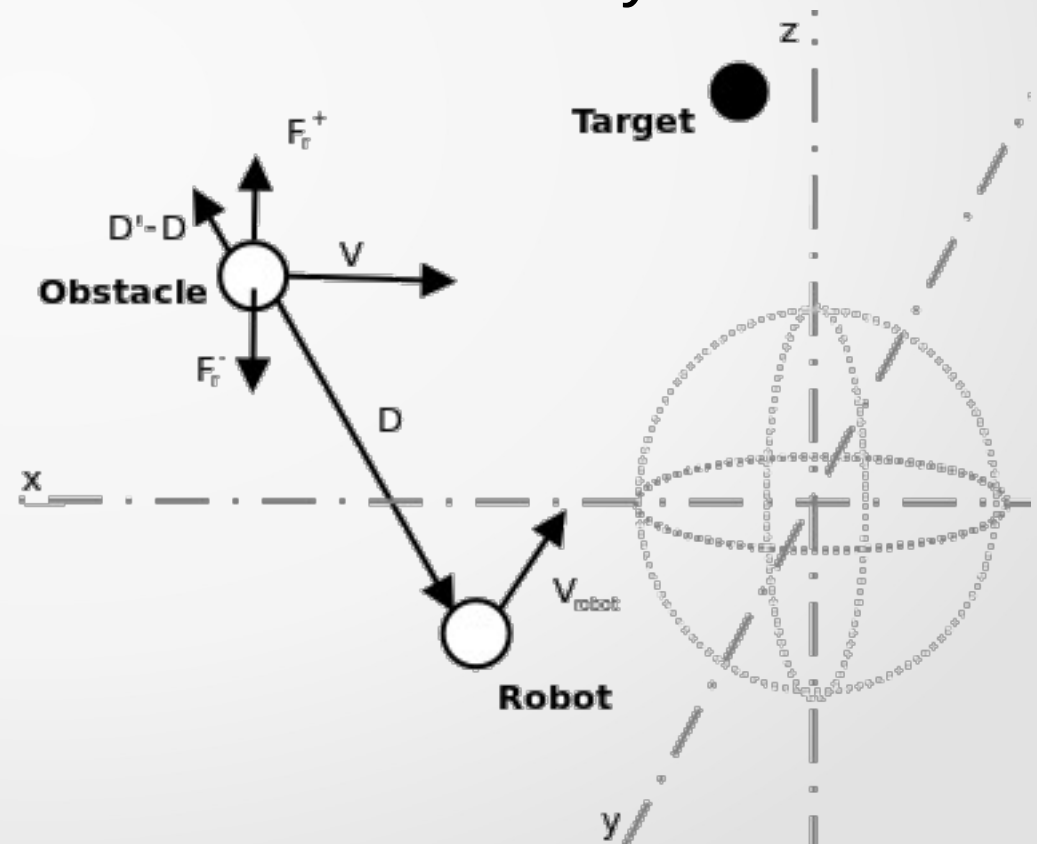
→ Virtual repulsion force

$$\vec{F}_r = \begin{cases} \vec{F}_r^+ & \text{if } \frac{F_r^+ - V_{robot}}{|\vec{F}_r^+| V_{robot}} > \frac{F_r^- - V_{robot}}{|\vec{F}_r^-| V_{robot}} \\ \vec{F}_r^- & \text{otherwise} \end{cases}$$

→ Force possibilities

$$\vec{F}_r^+ = (\vec{D}' - \vec{D}) \times \vec{V}$$

$$\vec{F}_r^- = \vec{V} \times (\vec{D}' - \vec{D})$$



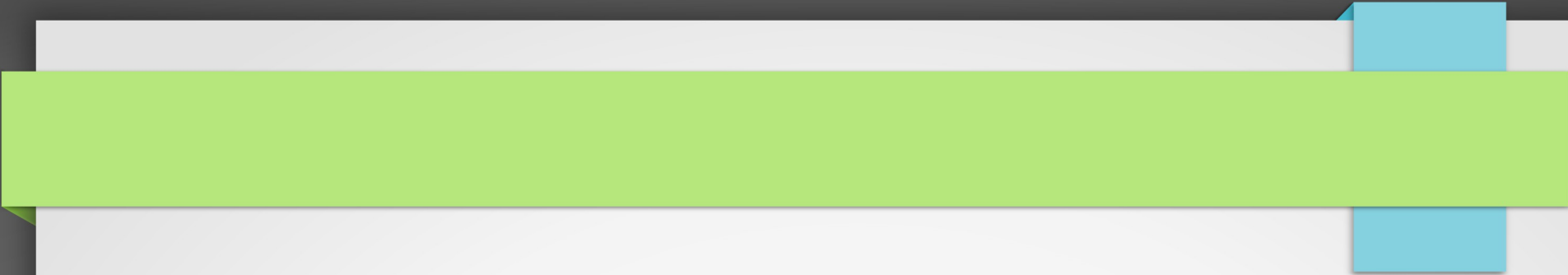
Conclusions

Accomplished goals

- Definition of the world model.
- Programming the world model.
- Identifying and defining possible scenarios.
- Programming each defined scenario.
- Modelling the calculated plan as a task sequence.

Further work

- Build the Visual Digester and the Pilot Nodes.
- Take into account the error margin between model and reality.
- Build the actual robot.



Thanks for your attention.