SOFTWARE DESIGN SPECIFICATIONS OF

Warehousing and Transporting: Multi-Robot Task Planning

Version 1.0

May 24, 2017

Advanced Agent Competition:
Human-Robot Collaboration and Teamwork (SS 2017)

Distribution Group:
Internal

Main responsible Coordinator:
DAI-Labor, TU-Berlin

Teammembers:
Jacob Seibert
Malte Siemers
Marc Schmidt

WAREHOUSING AND TRANSPORTING: MULTI-ROBOT TASK PLANNING

TABLE OF CONTENTS

1 Introduction

1.1 Motivation and Problem Definition

Industry 4.0 strives to create so called "smart factories". In these the whole production starting from a request up to the finished product is one integrated process. To accomplish this, the process is devided in automated, intelligent modules, i.e. planning, manufacturing, processing and storage. Our project aims to build a storage module that is able to operate an automated warehouse. This warehouse is made of storage units that are accessible by diffent types of robots. Packages arrive at input locatons, are picked up by a suitable robot and are then transported to a free storage unit. If a package is requested for shipping, a robot will bring it from the respective storage unit to an output location.
To solve this problem of automated warehousing we split it into several subproblems:

Task Planning handles input and output requests and coordinates unoccupied robots

Storage Management handles the information on free and used storage units.

Trajectory Planning is used by a robot to determine the shortest path on a given road map, i.e. a graph in which the edges are "streets" without overlap in the configuration space of a robot.

Motion Planning provides a local motion model for the robots that lets them drive on given paths and handles dynamic obstacles.

1.2 Objectives

To solve the aforementioned problem, we want to:

- Build a simulated warehouse in the MORSE environment.
- Generate a roadmap using the map from MORSE
- Implement an input/output generator that accesses the data from the image recognition project.
- Implement Task Planning, Storage Management, Trajectory Planning and Motion Planning as a set of interacting ROS nodes.
- Connect ROS and MORSE to simulate a fully automated warehouse.

1.3 General Constraints

We work under the following assumptions:

- The set of possibly incoming packages anlong with weight and shape of them are known.
- The set of robots in action is known along with their configuration and specification, i.e. geometry of the robot, its turning radius and maximum load capacity.
- Storage units are big enough to handle all kinds of packages.
- Robots know their exact position

2 State-of-the-Art

In section 1.1 we decomposed the problem of automated warehousing into several subproblems. The storage management is a simple model that listens to a set of sensors and manages this information. The other parts, i.e. Task Planning, Trajectory Planning and Motion Planning, are active fields of research. Task planning is the process of identifying tasks and assigning them to available and able agents in an optimized fashion. Yan et al.[15] devide this problem into task decomposition and task allocation. The former tries to decompose a task into atomic units while the latter assigns those units to agents. Task allocation has a wide range of possible approaches. A market-based approach by Michael et al.[8] needs all robots to know about all possible tasks and bid for them based on a possible benefit. Shiroma and Campos[12] proposed a framework that lets robots run multiple tasks at once, choosing the best according to some regularly updated criteria and trade tasks that are less attractive to do with other robots for better ones. A hierarchial planner using symbolic logic to accomplish tasks was discribed by Marthi, Russel and Wolfe[7].
Trajectory Planning in general describes the problem of finding optimal paths in a known or unknown environment. In a warehouse a map exists and so the remaining problem is to find shortest paths on this map. Peasgood et al.[9] address this problem by generating collision free roadmaps that define fixed paths for robots. Rapidly exploring Random Trees (RRTs) can be used to find a fast but not optimal solution. The basic idea was improved by Karaman et al.[5]. Those approaches do not take dynamic environments into account. Moving obstacles can be modeled by generating a potential field based on local sensor data. An optimal path is than the one with minimal work along it and can be found with variational calculus[13]. Van den Berg et al.[14] combined the roadmap approach with a second local search for optimal trajectories along the edges of the roadmap graph.
We decided to take a synergetic approach towards these problems. Our Taskplanner hierarchially develops plans and ranks those plans by their approximated execution time. Our Trajectory and Motion Planning is a hybrid solution using roadmaps with a local planner to avoid moving obstacles.

3 Requirements

We clearly define the following points to be realized in this project:

- The warehousing system shall be visualized using the MORSE simulator [3].
- The environment map including definition of positions of input, output and storage shall be exchangeable and separately stored in file(s).
- The warehouse environment shall be equipped with sensors to detect packages and shall be modeled by multiple ROS [10] nodes.
- For the sensors and actuators existing models from MORSE [3] shall be used.
- It shall be possible to have multiple inputs and/or outputs in a warehouse map.
- The number of robots used shall be a customizable parameter of the system.
- At least two different robot types with different capabilities shall be available. Already existing MORSE [3] models of robots shall be used.
- At least two different package types shall be available.
- System messages (information, errors) shall be published via the ROS [10] logging mechanism.
- The input/output generator shall be exchangeable to allow embedding the warehousing system in an Industry 4.0 context. An interface for input and output of packages shall be available.

4 Scenarios

The specified system is located at the end of an Industry 4.0 manufacturing process. Products ordered by customers are produced individually and on request. These products are inspected afterwards to reject deficient products. This step is done cooperatively by humans and robots to exploit their particular strengths. Accepted products are packaged and the packets are then stored in a warehouse until they are requested to be shipped. Figure 1 depicts the overall manufacturing pipeline and the location of the warehousing system.

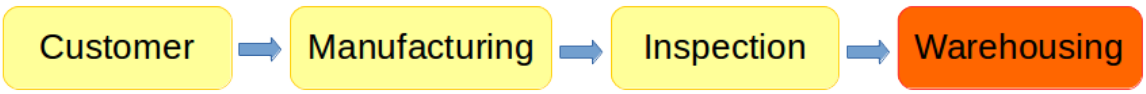

Figure 1: Location of the warehousing subsystem in the Industry 4.0 manufacturing pipeline.

4.1 Use cases

Looking at the system from the outside we have two main use cases which also define the interface of the warehousing system to the environment.

Package input Whenever a produced product has passed the inspection and packaging process it arrives at an input slot of the warehousing system. Together with the package some information about it are fed into the warehousing system. These informations contain an unique identifier of the package as well as it's weight and size. The task planner component creates a job and assigns it to an available robot. The robot then carries the package from the input slot to a free warehouse tray.

Internally, this use case consists of several steps such as robot and tray selection, transportation, loading and unloading. This use case is triggered by other components of the factory, e.g. the output of the packaging process.

Package output At any time a specific package stored in the warehouse can be requested at the output. The unique identifier of the package must be provided to retrieve the package. A robot then carries the requested package from the corresponding warehouse tray to one of the output slots.

This use case is triggered by software. This can for example be a delivery scheduler that takes desired delivery dates from the customer and requests the packages from the warehouse when they should be delivered.

4.2 Misuse cases

As the use cases incorporate many different tasks of multiple agents there are also many misuse cases which describe malfunction of the system that has to be handled in an appropriate way.

Load or unload fail If loading or unloading of the package by a robot fails, the package is neither correctly loaded on the robot nor placed in a tray. This can be detected by the discrepancy between sensors on the robot and in the tray. In this case the situation of the package is unknown, therefore it is not possible to load the package again. The system informs a human worker to pick up the package by hand and the robot is free to take new jobs.

No robot available When a new job is to be assigned to a robot but no robot is available, the job has to be suspended and assigned again after a certain time. The robots can be unavailable either because they are currently busy with another job, because they have a low battery and are charging or because they are not capable of carrying the package which depends on it's weight and size.

No free tray Creating a new job can fail because there is no free target warehouse tray or output slot. The job then has to be suspended and checked again after a certain time. This can lead to a temporarily blocked input slot.

Unmanageable package If there is a package with a weight or size such that it cannot be carried by any of the robots in the fleet, the input slot would be blocked forever. The system is not designed to handle this problem and it is assumed that only packages which can be carried by at least one robot arrive at the inputs.

Request of inexistent package If a package is requested which is not stored in the warehouse (either because it didn't arrive at the input or it is still carried by an robot and not yet stored in a tray) an error message is generated to inform the user that the request failed.

Robot breaks down Robots could possibly break down due to several reasons, e.g. a low battery. As far as such a failure can be detected by the system (battery sensor) a human worker is informed to remove the robot.

5 System Architecture and Specification

5.1 System Architecture

A visualisation of our system architecture is provided in Figure 2. It shows the connections between the different system components that will be implemented as ROS [10] nodes. The data flow is represented by the arrows. Since the agents are the part of the system doing the most extensive computation work, their operations are presented there too.

5.2 System Specification

Warehouse Management

The Warehouse Management has access to all the data, that is required to obtain a certain model of a warehouse. This static data is used to describe the physics of the system as well as to configure the different components of it as required. The Warehouse Management is responsible for getting the whole system started with the predefined configurations and to stop the incoming of new tasks to shutdown the system if that is demanded.

- Data
  - Map - one map per simulation only
    - geometry of the environment
    - locations of input, output & storage
    - charging zones: rectangle locations in the x-y coordinate plane with preset charging rates
    - start / rest locations for robots: robots rest there if they are idle
  - Package Config - contains as many different package configurations as different packages needed

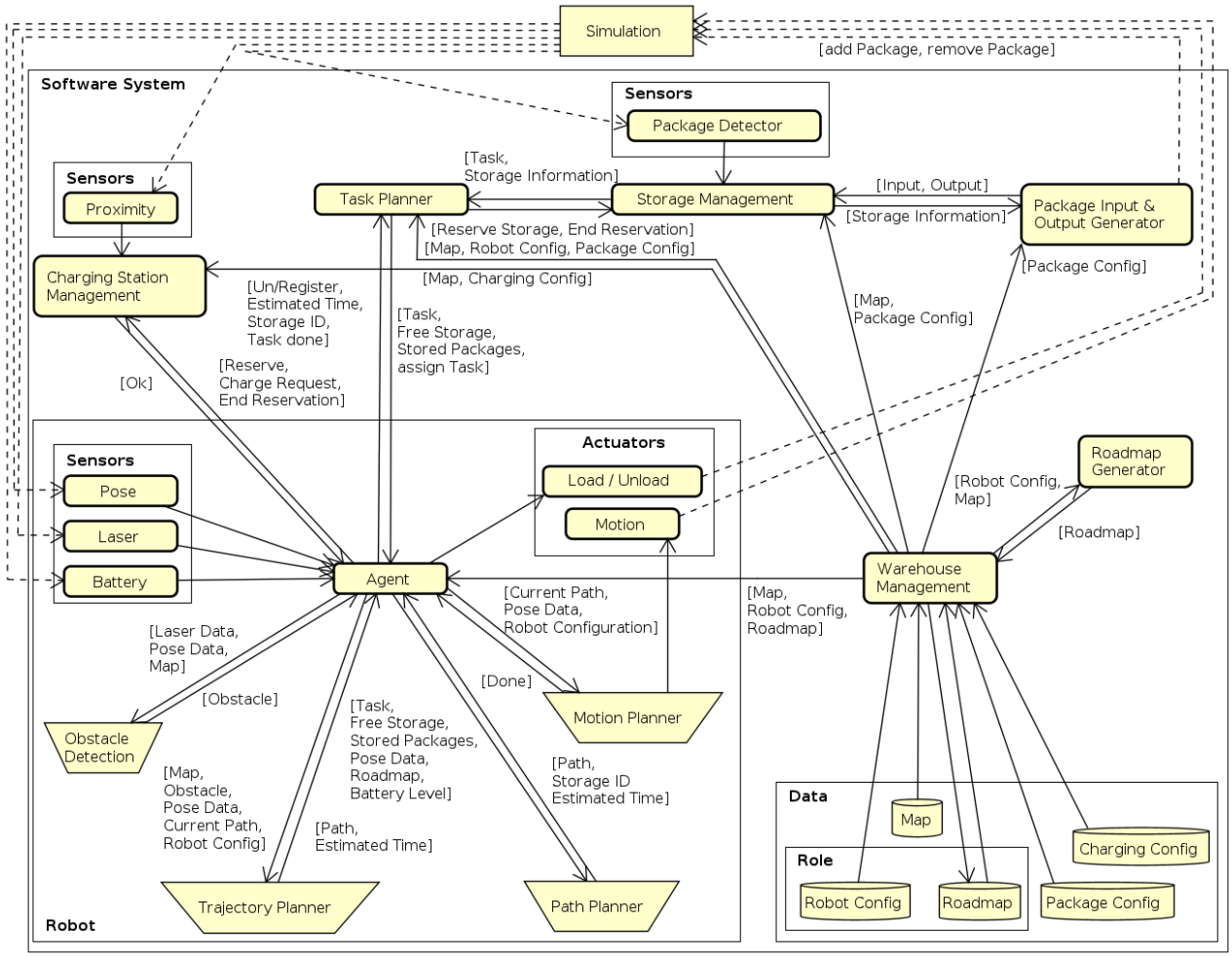

Figure 2: The overall System Architecture

  - Package_Type: all packages of a type are considered as equal, so if a package is requested at the output the software system only needs to care about the type of that package
  - Package_Weight: the weight of the package determines which robots are able to carry that package
  - Package_Geometrie: the geometry of the package determines which robots are able to carry that package
- Robot Config - contains as many robot configurations as different robot types needed
  Since we use only robots with differential drive the robots are capable of turning on spot so that we don't need to set a minimal turning radius in tthe robot configurations.
  - Max_Speed: the maximal velocity the robot is able to drive with
  - Robot_Radius: minimal radius of a circle centered in the middle of the robot, such that the circle covers the robot extent
  - Max_Carry_Weight: the maximal weight the robot is able to carry
  - Carry_Geometries: a list of all the geometries the robot is able to carry

        * Discharging_Rate: battery discharging rate in percent per second
    - Charging Config - contains one charging configuration per charging zone
        * Max_Number_Robots: the maximal number of robots, that can be charged simultaneously
    - Roadmap - one roadmap per robot configuration
        * is generated while running time by the Roadmap Generator
        * a static set of roads the robot is able to drive on
        * is represented as a graph: the nodes are associated with the different waypoint positions on the map, the edges specify which waypoints are directly reachable (in a straight line) from a certain waypoint and the edge weights contain the estimated time a robot would take to drive from one of this edge defining nodes to the other

- Service Requests
    - Roadmap Generator
        * generateRoadmap(Map, Robot_Config) : Roadmap
            · at the beginning of the simulation the Warehouse Manager requests the Roadmap Generator to create one Roadmap per robot configuration
            · Roadmaps are required to start the agents because they contain the data to plan the path for the robot
    - Package Input & Output Generator
        * start(Package_Config) : boolean
            · after all other components have been launched, finally the Package Input & Output Generator has to be started to simulate requests to store or retrieve packages
        * stop() : boolean
    - Storage Management
        * start(Map, Package_Config) : boolean
    - Task Planner
        * start(Map, Robot_Config, Package_Config) : boolean
    - Agent
        * start(Agent_ID, Map, Robot_Config, Roadmap) : boolean
    - Charging Station Management
        * start(Map, Charging_Config) : boolean

Roadmap Generator

The Roadmap Generator offers a service to create roads on a static map with respect to the physical constraints of the vehicles that are about to use this roads.

- Service Offers
    - generateRoadmap(Map, Robot_Config) : Roadmap
        * generates a Roadmap with respect to the static obstacles the Map contains as well as the Robot_Config which determines which movement behaviour the specific kind of robot has
        * the Map contains important locations the roads have to cover e.g. input / output & storage locations, charging zones, start / rest locations, ...

    * the algorithm we are going to use for generating the roadmaps is presented in [1]

Package Input & Output Generator

The Package Input & Output Generator simulates a source such that packages arrive at the input for storage in the warehouse as well as it simulates a sink where a specific type of package is demanded at some given output and the system shall provide that package there as soon as possible. The generation of requests shall be random.

- Service Offers
  - start(Package_Config) : boolean
    * hands over the package configurations
    * starts to simulate package sources as well as package sinks
    * requests storage information from the Storage Management
  - stop() : boolean
    * stops to create new packages at the inputs as well as it stops to demand stored packages at the outputs
- Service Requests
  - Storage Management
    * getStorageInformation() : Storage_Information
      · directly after being started, the Package Input & Output Generator asks for the storage informations, to get the required informations for generating inputs & outputs
    * newInput(Input_ID, Package_ID, Package_Type) : boolean
    * newOutput(Output_ID, Package_Type) : boolean
- Topics Subscribed
  - Storage Management
    * Storage Information
      · used to stay up to date with the storage informations
      · informations are published when the state of the warehouse storage changed
      · required to know where to add input and which output to request

Storage Management

The Storage Management has an overview of the overall storage of the warehouse and provides those information, performs administrative tasks like reservation of storage and informs the Taskplanner about incoming Tasks. Furthermore the Storage Management is able to detect if a package is stored at a certain storage location so that loading as well as unloading packages is doublechecked.

- Service Offers
  - start(Map, Package_Config) : boolean
    * hands over the map as well as the package configurations
    * starts all other services and publishes the storage informations topic
  - getStorageInformation() : Storage_Information

    - provides the storage informations of the whole warehouse
    - useful at start because the storage information topic only publishes if the package detectors recognized some change
  - newInput(Input_ID, Package_ID, Package_Type) : boolean
    - a new package with a given id and type has been arrived at the input container with the given id
    - saves the id and type of the package corresponding to the input container with the given id
    - expects the package detection laser to recognize an incoming package at the input with the given id and marks that input as reserved
  - newOutput(Output_ID, Package_Type) : boolean
    - a package of the given type is demanded at the output container with the given id
    - saves the type of the package corresponding to the output container with the given id
    - expects the package detection laser to recognize an incoming package at the output with the given id and marks that output as reserved
  - reserveStorage(Storage_ID) : boolean
    - questions to reserve the storage with the given id
    - reserved storage is not marked as free storage
    - returns if the storage is reserved
  - endReservation(Storage_ID) : boolean
    - questions to end reservation of the storage with the given id
    - the storage is marked as free again
    - returns if the reservation has been ended

- Topics Published
  - Storage Information
    Every time a package has been placed into or has been retrieved from a storage container and every time a storage container is reserved or not any longer reserved the Storage Informations are published.
    - Free_Input: Input_ID's of the input containers that are empty
    - Free_Output: Output_ID's of the output containers that are empty
    - Free_Storage: Storage_ID's of the storage containers that are empty
    - Stored_Packages: information about where and which packages are stored in the warehouse e.g. Storage_ID, Package_ID, Package_Type,...
    - Full_Output: Output_ID's of the output containers that are full

- Service Requests
  - Task Planner
    - storePackage(Input_ID, Package_ID, Package_Type) : boolean
      - after getting the new input request and detecting some package at the corresponding input the task is sent to the planner
    - retrievePackage(Output_ID, Package_Type) : boolean
      - after getting the new output request and making sure such a package type is stored in the warehouse already the task is sent to the planner

- Topics Subscribed
  - Package Detector Sensor - one per input / output & storage container on the map
    - Package_Present
      - detects whether a package is present at the container or not
      - could be handled by collision detection or laser detection

Task Planner

The Task Planner receives tasks, decides which kind of robot is capable to fulfill those, requests how much time it would take for the suitable robots to accomplish the task, decides which robot is the most available one and assigns the task to it.

- Service Offers
  - start(Robot_Config, Package_Config) : boolean
    - hands over the map, the robot configurations and the package configurations
    - starts all other services
  - storePackage(Input_ID, Package_ID, Package_Type) : boolean
    - a package with the given id and type arrived at the input with the given id and is ready to be stored in the warehouse
    - chooses which of the registered robots are capable of fulfilling the task and sends the store request to them
  - retrievePackage(Output_ID, Package_Type) : boolean
    - a package with a given type is demanded at the input with the given id
    - chooses which of the registered robots are capable of fulfilling the task and sends the retrieve request to them
  - register(Agent_ID) : boolean
    - registers the agent with the given id
    - registered agents receive task requests
  - unregister(Agent_ID) : boolean
    - unregisters an agent to avoid receiving task requests
    - will be called if the battery of an Agent is below some minimum and it's not clear if the Charging Station is going to be reached before it's empty
  - completedTask(Task_ID) : boolean
    - called by an agent if the task with the given id has been fulfilled
    - when the Storage Management provides the expected Storage Information the task is completed
- Service Requests
  - Agent
    - store(Task_ID, Input_ID, Free_Storage) : Storage_ID, Estimated_Time
    - retrieve(Task_ID, Output_ID, Stored_Packages) : Storage_ID, Estimated_Time
    - assignTask(Task_ID) : boolean
      - choose agent with the lowest estimated time for that task

            - before calling this tell the Storage Management must have reserved the corresponding storage
    - Storage Management
        - reserveStorage(Storage_ID) : boolean
            - requests to reserve storage if an agent asked to do so
        - endReservation(Storage_ID) : boolean
            - requests to end reservation if an agent asked to do so or if the associated task is complete

- Topics Subscribed
    - Storage Management
        - Storage Information
            - needs storage informations to tell the agents which possibilities they have to store the package at or retrieve the package from

Agent

The Agents are the executive components of the whole system. They receive requests to calculate how much time they would require to fulfill a specific task and have to take care of those tasks. Doing this the battery level is considered to include visiting charging zones which should be integrated into the calculation of the duration the task could be fulfilled in. The Agents have access to a pose, laser and battery sensor to get information about their location and their local environment as well as they have access to load / unload and motion actuators to interact with their environment.

- Operations
    - detectObstacles(Laser_Data, Pose_Data, Map) : Obstacle
        - if the laser data doesn't match the data of the static obstacles on the map the unexpected obstacle is returned
        - an obstacle consists of a set of points in the x-y coordinate plane
        - no complex algorithm needed, just match the laser data with the static obstacles
        - returns NULL if no Obstacle detected
    - planPathToStorage (Input_ID, Free_Storage, Pose_Data, Roadmap, Battery_Level) : Path, Storage_ID, Estimated_Time
        - calculates the shortest path on the roadmap that is required to store the package at the input with the given id
        - considering the battery level it takes the visit of charging stations into account in the computation
        - chooses one of the empty storage containers to store the package at
        - start point of the path is the destination of the last task in the taskqueue of the robot and the battery level is the estimated battery level at this location
        - shortest path finding with dijkstra
    - planPathToDeliver (Output_ID, Package_Type, Stored_Packages, Pose_Data, Roadmap) : Path, Storage_ID, Estimated_Time
        - calculates the shortest path on the roadmap that is required to deliver a stored package of the given type to the output with the given id

    - considering the battery level it takes the visit of charging stations into account in the computation
    - chooses one of the containers that contain the given package type to retrieve the package from
    - start point of the path is the destination of the last task in the taskqueue of the robot and the battery level is the estimated battery level at this location
  - planTrajectory
    (Map, Obstacle, Pose_Data, Current_Path, Robot_Config) : Path, Estimated_Time
    - is triggered when an obstacle is detected
    - calculates the shortest path back to the current path considering the given obstacle, the map and the robot configuration
    - may skip some of the waypoints of the current path but may not skip subgoals e.g. storage location to load / unload a package
    - start point of the returned path is the current location of the robot and the destination point of the returned path is some point of the current path
    - use the D* algorithm here
  - planMotion(Current_Path, Pose_Data, Robot_Configuration) : boolean
    - adjusts the parameters of the motion actuator to drive along the given current path and takes into account the current pose data as well as the robot configuration
    - tells motion actuator which linear and angular velocity to set
    - returns true if already at destination

- Service Offers
  - start(Agent_ID, Map, Robot_Config, Roadmap) : boolean
    - hands over the agent id, the map, the robot configuration and the roadmap associated to the robot configuration
    - starts all other services
    - calls the register service at the task planner
  - store(Task_ID, Input_ID, Free_Storage) : Storage_ID, Estimated_Time
    - request to plan the storing of a package which is located at the input with the given id
    - returns the estimated time for this task in addition to the time for completing all other tasks in the taskqueue and returns the id of the storage the package should be stored at
    - calls planPathToStore with the associated parameters
    - saves task as well as calculation results temporarily
  - retrieve(Task_ID, Output_ID, Stored_Packages) : Storage_ID, Estimated_Time
    - request to plan the delivering of a package to the output with the given id
    - returns the estimated time for this task in addition to the time for completing all other tasks in the taskqueue and returns the id of the storage the package will be retrieved from
    - calls planPathToDeliver with the associated parameters
    - saves task as well as calculation results temporarily
  - assignTask(Task_ID) : boolean
    - assigns the task with the given id to this agent
    - the task is placed at the last position of the taskqueue of the agent
    - we know which task to fulfill because we saved it temporarily before

    - calls reserveChargingZone

- Service Requests
  - Charging Station Management
    - reserveChargingZone(Agent_ID, ChargingZone_ID) : boolean
      - will be called after a task is assigned and it requires charging to fulfill the task
    - chargeRequest(Agent_ID, ChargingZone_ID) : boolean
      - triggered a short time before the robot wants to charge
    - endReservation(Agent_ID, ChargingZone_ID) : boolean
      - triggered after succesful charging or if charging plans change
  - Motion Actuator - one per agent
    - stop() : boolean
      - stops the robot from moving
      - sets linear and angular velocity to 0
    - setSpeed(Linear_Velo, Angular_Velo) : boolean
      - sets the linear and angular velocity of the motion actuator allowing the robot to move in the x-y coordinate plane
  - Load / Unload Actuator - one per agent
    - load() : boolean
      - grips a package in front of the robot and attach it to itself
      - no need to arrange the arm posure, just take it
    - unload() : boolean
      - detaches the attached packet and puts it in front of the robot
      - no need to arrange the arm posure, just release it

- Topics Subscribed
  - Pose Sensor - one per agent
    - offers the current Position of the robot as well as the direction the robot is heading to
  - Laser Sensor - one per agent
    - offers the list of positions of the points the lasers are pointing to
    - positions are returned as triple (x,y,z)
    - if there is no abstacle in front of a laser it returns (0,0,0) for that laser
  - Battery Sensor - one per agent
    - offers the battery level in percent

Charging Station Management

The Charging Station Management controls and administrates the charging zones. If a robot wants to load his battery, it needs to ask the Charging Station Management if that is possible.

- Service Offers
  - start(Map, Charging_Config) : boolean
    - hands over the map and the charging configurations

    - starts all other services
  - reserveChargingZone(Agent_ID, ChargingZone_ID) : boolean
    - if the maximum number of reservations in the charging zone with the given id hasn't been reached yet the agent with the given id gets a reservation
    - otherwise the reservation request is added to a waiting queue
  - chargeRequest(Agent_ID, ChargingZone_ID) : boolean
    - returns true if the agent with the given id has a reservation at the charging zone with the given id and the proximity sensors data tells that there are only robots with reservation in the charging zone
    - returns false if the agent's reservation request is in the waiting queue or if there are are more agents in the charging zone than allowed
  - endReservation(Agent_ID, ChargingZone_ID) : boolean
    - terminates the reservation of the agent with the given id at the charging zone with the given id
    - removes reservation request from waiting queue

- Service Requests
  - Proximity Sensor - one per charging zone
    - getAgents() : [Agent]
      - returns a set of agents located in the radius of the proximity sensor

Simulation

The simulation done by MORSE [3] simulates a real-world warehouse scenario. All sensors that have been mentioned before get their sensor data from the simulation as well as all actuators have a direct impact to the simulation, so that the data received by the sensors changes over time. In addition to the actuators the Package Input & Output Generator is able to affect the Simulation environment while it creates & places or removes packages while runtime to provide a realistic scenario.

6 Timeline

W6
- test sensors & actuators
- setup a simple map & robot configurations
- implement Warehouse Management
- implement dummy Taskplanner
- setup Agents with pose sensor & motion actuator
- decide which Roadmap Generator algorithm to use

W7
- add & integrate load / unload actuator
- start to implement the Roadmap Generator & Path Planner
- implement the Motion Planner

W8
- finish Roadmap Generator & Path Planner
- test Motion planner and path planner in MORSE [3]

Milestone I - W9 (13.06)

- test whether MORSE [3] sensors & actuators perform as demanded, if some don't their behaviors could be simulated at the expense of the real-world subscription of the MORSE Simulation [3]
- create a simple warehouse map & at least two different robot configurations
- implement the Warehouse Management
- implement the Roadmap Generator - should be able to generate a roadmap for any map and any robot configuration
- add a dummy Task Planner - tells the robots to which position to drive
- setup the Agent
  - add & integrate Pose Sensor & Motion Actuator - robot should be able to drive to to the given location
  - add & integrate Load / Unload Actuator - robot should be able to grab, transport & release packages
  - implement Path Planner - find the shortest path on the roadmap & calculate the estimated time for driving along this path
  - implement Motion Planner - robot should be able to drive along a given path (if there's not enough time we could use some motion actuator that drives to the next waypoint in a straight line instead of calculating the linear and angular velocity)

W9
- create a complexer map with charging zones
- create package onfigurations
- implement Package Input & Output Generator
- implement Storage Management
- change & integrate Task Planner
- decide which Trajectory Planner algorithm to implement

W10
- create charging configurations
- implement Charging Station Management
- expand Agent with Laser & Battery Sensor
- integrate charging into Path Planner
- start to implement the Trajectory Planner

W11
- improve Motion Planner
- finish the Trajectory Planner
- couple Path Planner and Trajectory Planner
- add moving obstacles to MORSE / add more robot instances

W12
- final tests and finalization of the MORSE integration
- final debugging

Milestone II - W13 (11.07)

- create a complexer map & add at least two charging zones to it
- create at least two different package configurations as well as charging configurations
- implement Package Input & Output Generator - add mechanism to create & remove packages in the simulation
- implement Storage Management and integrate package detection sensors
- implement Charging Station Management and integrate proximity sensors
- change & integrate the Taskplanner
- expand Agent
  - add & integrate Laser Sensor
  - add & integrate Battery Sensor
  - implement Trajectory Planner
  - integrate charging into Path Planner
  - improve Motion Planner
- setup all the communications between the components
- robots shall be able to fulfill incoming tasks while preventing their battery from discharging as long as possible

7 Framework and Tools

For the simulation and visualization the MORSE simulator [3] will be used. The system itself will be implemented using ROS [10]. Individual ROS nodes will be implemented using either C++ or Python.

The roadmap for each robot type will be generated using the method proposed in [1]. Path planning in the road map will be done using Dijkstra's famous algorithm [2]. For the local planner existing ROS solutions will be used. Possible packages are `base_local_planner` [4] or `teb_local_planner` [11].

For the task planner we will use a hierarchical task network (HTN) planning approach similar to [6].

References

[1] Valerio Digani, Lorenzo Sabattini, Cristian Secchi, and Cesare Fantuzzi. An automatic approach for the generation of the roadmap for multi-agv systems in an industrial environment. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 1736–1741. IEEE, 2014.

[2] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[3] Gilberto Echeverria, Severin Lemaignan, Arnaud Degroote, Simon Lacroix, Michael Karg, Pierrick Koch, Charles Lesire, and Serge Stinckwich. Simulating Complex Robotic Scenarios with MORSE. In *SIMPAR*, pages 197–208, 2012. MORSE Simulator Documentation: https://www.openrobots.org/morse/doc/stable/morse.html; 22.05.2017.

[4] Eric Perko Eitan Marder-Eppstein. base_local_planner. http://wiki.ros.org/base_local_planner. [Online; accessed 23-May-2017].

[5] Sertac Karaman, Matthew R Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. Anytime motion planning using the rrt. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1478–1483. IEEE, 2011.

[6] Raphaël Lallement, Lavindra De Silva, and Rachid Alami. Hatp: An htn planner for robotics. *arXiv preprint arXiv:1405.5345*, 2014.

[7] Bhaskara Marthi, Stuart J Russell, and Jason Wolfe. Angelic semantics for high-level actions. In *ICAPS*, pages 232–239, 2007.

[8] Nathan Michael, Michael M Zavlanos, Vijay Kumar, and George J Pappas. Distributed multi-robot task assignment and formation control. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 128–133. IEEE, 2008.

[9] Mike Peasgood, Christopher Michael Clark, and John McPhee. A complete and scalable strategy for coordinating multiple robots within roadmaps. *IEEE Transactions on Robotics*, 24(2):283–292, 2008.

[10] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*, 2009. Download Paper: http://www.willowgarage.com/sites/default/files/icraoss09-ROS.pdf, ROS Webpage: http://www.ros.org/; 22.05.2017.

[11] Christoph Rösmann. teb_local_planner. http://wiki.ros.org/teb_local_planner. [Online; accessed 23-May-2017].

[12] Pedro M Shiroma and Mario FM Campos. Comutar: A framework for multi-robot coordination and task allocation. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 4817–4824. IEEE, 2009.

[13] Mikael Svenstrup, Thomas Bak, and Hans Jørgen Andersen. Trajectory planning for robots in dynamic human environments. In *Intelligent robots and systems (IROS), 2010 IEEE/RSJ international conference on*, pages 4293–4298. IEEE, 2010.

[14] Jur P Van Den Berg and Mark H Overmars. Roadmap-based motion planning in dynamic environments. *IEEE Transactions on Robotics*, 21(5):885–897, 2005.

[15] Zhi Yan, Nicolas Jouandeau, and Arab Ali Cherif. Multi-robot decentralized exploration using a trade-based approach. In *ICINCO (2)*, pages 99–105, 2011.