DEPENDABLE EDGE COMPUTING

FOR FLEET MANAGEMENT SYSTEM

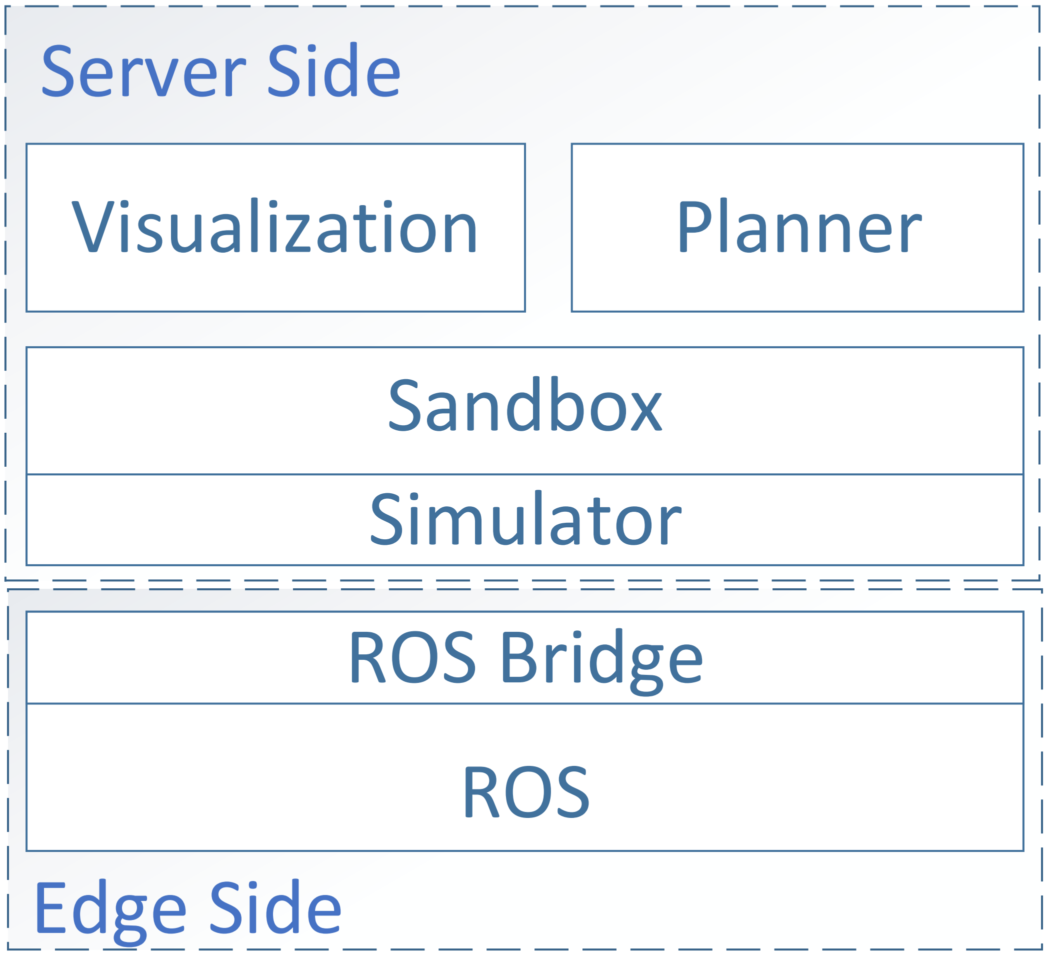
Software Pointer

Version	Time	Description
Alpha	2019-1-2	Initial version.

CONTENTS

OVERVIEW OF SOFTWARE

The software is designed for fleet management, currently focus on monitoring and rebuilding the running environment of fleet. Currently the system includes four subsystems, which indicated as following figure. In this system, we assume low level sensor fusion has been done on the edge side, i.e. only high-level environment info will be sent to server side and processed.



SETUP AND RUN

PREREQUEST

- Ubuntu 16.04 LTS
- Python 3.6
- OSMnx: https://github.com/gboeing/osmnx.git
- UTM: https://github.com/Turbo87/utm.git
- RDFLib: https://rdflib.readthedocs.io/en/stable/
- Scott-eu/planner_resoner: https://github.com/EricssonResearch/scott-eu.git
- Docker
- Git

RUN CURRENT SOFTWARE

1. Clone project repository:
   ```
   $ git clone https://github.com/XTC0908/Dependable-Edge-computing.git
   ```

2. Clone Scott-eu repository:
   ```
   $ git clone https://github.com/EricssonResearch/scott-eu.git
   ```

3. Modify network setting:
   - Navigate to Dependable-Edge-computing/Sandbox
   - In *Simulator.py* line 15
   - If run for local test, set TCP_IP to *"127.0.0.1"*
   - If use simulator as bridge, set TCP_IP to *<ip_of_ros_node>*

4. Lunch subsystems:
   - Navigate to scott-eu/planner_reasoner, follow README instruction to lunch **planner**
   - Navigate to Dependable-Edge-computing/Viz, lunch Viz_map.py for **visualization**
   - Navigate to Dependable-Edge-computing/Sandbox, lunch sandbox.py for **Sandbox**

5. Lunch simulator for local test:
   - Navigate to Dependable-Edge-computing/Sandbox
   - Lunch simulator.py with following instruction and argument
     ```
     $ python simulator.py local 1 i jam
     ```
   - You can add another vehicle
     ```
     $ python simulator.py local 2 s smooth
     ```

6. Lunch simulator as a bridge to ROS Node:

- Navigate to Dependable-Edge-computing/rosws, follow instruction to setup ROS environment and lunch a ROS node.
- Navigate to Dependable-Edge-computing/Sandbox
- Lunch simulator.py with following instruction and argument

```
$ python simulator.py remote 1 i smooth
```

MAP SUBSYSTEM TO CODE

Following table indicates the subsystem and corresponding top/main program file/directory:

Subsystem	File/Directory
ROS node/ROS-bridge	Dependable-Edge-computing/rosws
Sandbox	Dependable-Edge-computing/Sandbox/sandbox.py
Simulator	Dependable-Edge-computing/Sandbox/simulator.py
visualization	Dependable-Edge-computing/Viz/Viz_map.py
planner	scott-eu/planner_resoner

SANDBOX DEVELOPMENT POINT AND COMMENT

OVERVIEW

Sandbox is the subsystem where road and fleet item (Digital Twin) are rebuilt and monitored by high-level environment info. It is event driven and all the objects and their relationships are represented as RDF-graph. Map is represented as a graph as default in OSMNx. Currently two functions are implemented: **1) naive risk assessment 2) smooth/jam detection and re-planning.**

COMMUNICATION

Sandbox straightforwardly communicates with Simulator, Visualization and Planner. The communication based on HTTP protocol, currently only body of HTTP is used.

COMMUNICATION WITH SIMULATOR

Sandbox is a HTTP server, always listening POST requests from simulator, server is initialized in *sandbox.py Line 230* Two type of requests body will be processed:

Type	Request Body	Expected Response	Description

New	{
	"type":"new",
	"vid": vid,
	"dest": [lon, lat],
	"position": [lon, lat],
	"time_stamp": ts
	}
		A plan which constructed as a list of tuple of way point pairs in the response body.	Add a new vehicle digital twin to the sandbox.

Pos	{
	"type":"pos",
	"vid": vid,
	"pos": [lon, lat],
	"u": waypoint_id,
	"v": waypoint_id,
	"time_stamp": ts
	"sensor": jam/smooth,
	}
		None	Update an exist vehicle with vid. "u", "v" is the road which vehicle is currently on, represented as start and end waypoint in a graph (refer to OSMNx). "sensor" is current road statues.

COMMUNICATION WITH PLANNER

Sandbox communicate with planner as a client, body is a RDF/Turtle format string of current digital twins, an example is in *Dependable-Edge-computing/rdf_generator/test_out.ttl.* This RDF/Turtle format string can be generated automatically through *problem_generator* in sandbox, the generator is initialized in *Dependable-Edge-computing/Sandbox/sandbox.py Line 187.* **Note: each time the map is updated, the generator needs to be reloaded with current map as *sandbox.py Line 179.***

COMMUNICATION WITH VISUALIZATION SUBSYSTEM

Sandbox communicate with Visualization Subsystem as a client, body is a json string. The communication is initialized on *Dependable-Edge-computing/Sandbox/sandbox.py Line 34,* also refers to *Dependable-Edge-computing/Sandbox/interfaceToViz.py* for implementation details.

RISK ASSESSMENT

The implemented risk assessment function calculates the braking distance of each vehicle, generate a bounding box and detect if two bounding box is overlapping. The procedure starts from *Dependable-Edge-computing/Sandbox/sandbox.py Line 143,* and implemented in *Dependable-Edge-computing/ calc_riskzone.py.*

The Risk assessment is very naïve because of limitation of project time duration. It would be interesting if it can be replaced as a real risk assessment algorithm. Since the map has been constructed as a graph, and vehicles are also vertices on the graph, it might possible to implement a graph inference algorithm on it.

RE-PLANNING

Re-planning function is implemented by remove jam road (edge in the graph) and re-generate RDF/Turtle file in program_generator. Implementation start from on *Dependable-Edge-computing/Sandbox/sandbox.py Line 167.*

SANDBOX CLASS

Sandbox is implemented as a class, following attributes and methods is implemented:

ATTRIBUTES

1. **top:** the top entry of RDF graph saved in sandbox class.
2. **viz:** the entry to the visualization subsystem.
3. **planner:** the entry to the planner subsystem.
4. **problem_generator:** the entry to PDDL compatible RDF/Turtle file generator.

METHODS

1. **add_vehicle (self, vehicle):** add a vehicle into RDF graph.
   Param: vehicle: a json file which describe a vehicle, field of json file refer to *Dependable-Edge-computing/Sandbox/simulator.py Line 97*
2. **update_vehicle (self, vehicle):** update an existing vehicle, indexed by vid field in json file.
   Param: vehicle: a json file which describe a vehicle, field of json file refer to *Dependable-Edge-computing/Sandbox/simulator.py Line 97*
3. **load_map(self, map_path, folder):** load a map into the RDF graph.
   Param: map_path: string, path to the map file.
   Folder: string, folder where the map file is.

4. **on_query(self, pos):** check whether a given position on road (edge of a map graph).
   Param: pos: tuple **of position, (longitude, latitude)**
   Return: None: if pos does not on the road.
   Tuple (way point1 id, way point2 id): Two end of the edge.

USAGE

An example usage of Sandbox class can be found in *Dependable-Edge-computing/Sandbox/Server.py Line 38,* the code segment is indicated.

```
            try:
                req_type = json_body['type']
                if req_type == 'new':
                    plan = self.sandbox.add_vehicle(json_body)
                    self.wfile.write(json.dumps(plan).encode('utf-8'))
                elif req_type == 'pos':
                    self.sandbox.update_vehicle(json_body)
            except:
                pass
```

SIMULATOR

The simulator (*Dependable-Edge-computing/Sandbox/Simulator.py*) is constructed for local test as well as convert Web-Socket (ROS_bridge) to HTTP(Sandbox), for further development, this part should be replaced by a test program and a gateway program (protocol converter and handling multiple connection).

PLANNER

The planner for path planning is built based on a heuristic algorithm – "STRIPS", whose reference can be found in the report, including some basic templates.

Online solver for initial version test: PDDL Solver https://bitbucket.org/planning-researchers/cloud-solver

A semantic WEB descriptionRDF is used for dependability in the data transition, tutorial : https://www.w3.org/TR/turtle/

Method to convert rdf-type domain problem file, can be cound in Ericsson's research. https://github.com/EricssonResearch/scott-eu

A template in the folder "rdf_generator" in Github with comment:

```
        // namespace for the domain object naming, usually not need to change.
        @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
        @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
        @prefix oslc: <http://open-services.net/ns/core#> .
        @prefix sh: <http://www.w3.org/ns/shacl#> .
        @prefix pddl: <http://ontology.cf.ericsson.net/pddl/> .
        @prefix : <http://ontology.cf.ericsson.net/pddl_example/> .
```

```
:edge-
computing
              a pddl:Domain ;
              oslc:instanceShape pddl:DomainShape ;
              rdfs:label "edge-computing" ;
              pddl:type :waypoint ,
                        :vehicle ;
              pddl:constant :dest ;
              pddl:predicate :on ,
                             :visible ;
              pddl:function :moved ,
                            :total-moved ;
              pddl:action :move .
```
//define the type of domain and the predicate, function and type which is the same in PDDL syntax.

```
:waypoint
              a rdfs:Class ;
              rdfs:subClassOf pddl:PrimitiveType ;
              oslc:instanceShape pddl:PrimitiveTypeShape ;
              rdfs:label "waypoint" .
```
//the description of a object in the domain, like a declaration.

```
:move
          a rdfs:Class ;
          rdfs:subClassOf pddl:Action ;
          oslc:instanceShape pddl:ActionShape ;
          rdfs:label "move" ;
          pddl:parameter :move-b ,
                         :move-x ,
                         :move-y ;
          pddl:precondition [ a pddl:And ;

                              pddl:argument

                                         [ a :on ;
                                           :on-x :move-b ;
                                           :on-y :move-x
```

```
                                                  ] ,
                                                  [
                                                     a pddl:Or ;
                                                     pddl:argument
                                                     [
                                                        a :visible ;
                                                        :visible-x :move-x ;
                                                        :visible-y :move-y
                                                        ] ,
                                                        [
                                                          a :visible ;
                                                          :visible-x :move-y ;
                                                          :visible-y :move-x
                                                        ]
                                                      ]
```

//the description of an action in the domain, including the object that operates the action and the precondition.

```
  pddl:effect
  [ a
  pddl:And ;
                               pddl:argument [ a :on ;
                                               :on-x :move-b ;
                                               :on-y :move-y
                                             ] ,
                                             [ a pddl:Not ;
                                               pddl:argument [ a :on ;
                                                               :on-x :move-b ;
                                                               :on-y :move-x
                                                             ]
                                             ] ,

                                             [ a pddl:Increase ;
                                               pddl:parameter [ a :moved ;
                                                                :moved-m :move-b
                                                              ] ;
                                               pddl:argument 1
                                             ] ,
                                             [ a pddl:Increase ;
                                               pddl:parameter [ a :total-moved ] ;
                                               pddl:argument 1
                                             ] ,
                                             [ a pddl:When ;

                       pddl:parameter [     a pddl:And ;

                      pddl:argument [
                                    a pddl:Equality ;

                      pddl:left :move-y ;

                      pddl:right :dest

                       ] ];

                             pddl:argument [ a pddl:Not ;
                                            pddl:argument [ a :visible ;
                                                          :visible-x :move-x ;
                                                          :visible-y :move-y
                                                           ]
                                                          ]
                                                           ]
                                                          ] .
               // The effect of the action, also indicates a terminal state of the
               action. The action definition is the most important in the problem
               description.
```

ROS/ROS_BRIDGE

Follow the instruction in to Dependable-Edge-computing/rosws/README.md to setup and understand this part. For further development, you may want to use physical vehicle, then you need following info:

MESSAGE TYPE

Message type suggested: geographic_msgs/GeoPoint, geographic_msgs/RoutePath, geographic_msgs/RouteSegment, geographic_msgs/WayPoint

Message type used: See notes at the beginning of Dependable-Edge-computing/rosws/src/edge_info/vehicle.py

PACK MESSAGES

Define messages in src/<package name>/msg/

Modify src/<package name>/Cmakelist.txt: Add the name of newly created message in 'add_message_files()' around line 53.

catkin_make clean $ catkin_make

Modify client files: See pub_client.py, sub_client.py, edgeweb.html

CREATE CONNECTION

Local connection : in pub_client.py, sub_client.py, TCP_IP = '127.0.0.1'. In edgeweb.html, url : 'ws://localhost:9090'.

Connect from another computer: Check local IP address using 'ifconfig' command. Change TCP_IP/url to local IP address.

VISUALIZATION

Visualization Subsystem is an event driven GUI program based on **TKInter**, it communicates with sandbox as a HTTP server, refer to Communication with Visualization section and the comment in source code for more details.