Table of Contents

Table of figures

Software modules definition

Coordinator

Coordination between system’s modules (photo capture, structure from motion and graphical user interface), file storage and serving, data serving are taking place in this module:

- Sets an HTTP server with the specified endpoints to handle HTTP requests, including file uploading/downloading. Responsible for synchronous requests (from modules, to coordinator).
- Sets a WebSocket server which accepts connections on specified endpoints, one for each module. Asynchronous messages from server to clients are transmitted using this server.
- Alternates some of acquired data, if it is not feasible in lower level (e.g. transforming a file’s format to another)
- Wraps the system’s logic by translating requests/messages, into events, which define system’s current state.
- Uploads information to FiWARE ORION as per project’s needs.

Photo capture

This module’s main responsibility is to imprint the object of interest from different angles. To do so, a high resolution camera is mounted on the robotic arm’s tip. The robotic arm is ordered to move around the object, and the camera to take pictures in specific positions. Once the procedure is complete, the images and some if their metadata (e.g. position of the camera for each image) are later on fed into structure from motion module to create a 3D representation.

Collision avoidance must be taken into account (taking as obstacles both the object of interest and the specific environment, e.g. table’s protection sides).

Structure from motion

Once stereoscopic view of the object has been accomplished by photo capture, it is fed (with metadata) to image processing algorithms. Main object’s 3D reconstruction, color mapping, texturing, scaling, positioning take place in this step.

Graphical User Interface

Each module’s output is visualized in a human-friendly application:

- Images from photo capture.
- Mesh from structure from motion.
- Photo capture/structure from motion completion status.

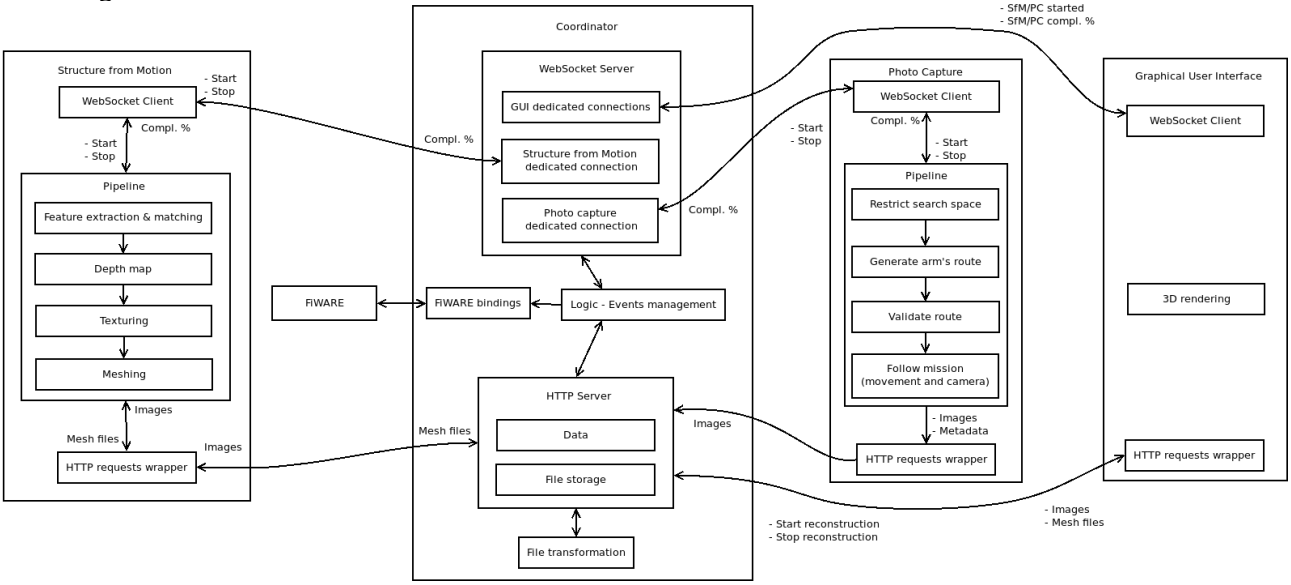
In addition, user input functionalities are encapsulated:

- Starting/Stopping photo capture.
- Starting/Stopping structure from motion.

Edge detection

Software modules implementation

Drawing 1: Architecture



System’s basic functionality can be summarized into the following steps:

1. The user clicks on “start” button (GUI)
2. Photo capture is triggered
3. Output images are fed to structure from motion
4. Output mesh is visualized

As per system’s modules definition, it consists of four top-level modules.

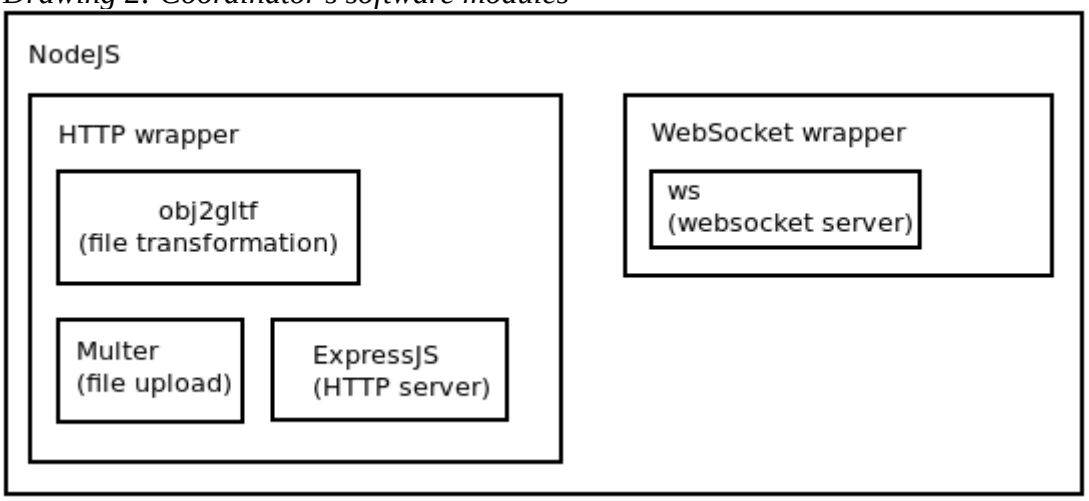
- Coordinator
- Photo capture
- Structure from motion
- Graphical user interface

Coordinator

A NodeJS based application, which:

- Sets the system’s main HTTP server using `expressJS`. Provides the following endpoints:
  - `/serve_image` : takes as query parameter image’s name and serves it for download.
  - `/cache_images` : expects files (photo capture’s images) in ‘files’ field, as a post form.
  - `/cache_mesh` : expects files (structure from motion mesh files) in ‘files’ field, as a post form. In addition to serving, “.obj” assets are converted to “.gltf”, since the latter has no need for a matching materials template file.
  - `/image_names` : returns stored images’ names.
  - `/start_photo_capture` : A “start” signal to photo capture module is transmitted once this endpoint is “visited” (HTTP GET).
  - `/stop_sfm` : A “stop” signal to structure from motion module is transmitted once this endpoint is “visited” (HTTP GET).
- Establishes a WebSocket server:
  - Clients are separated function-wise by their initialization’s endpoint:
    - `/ui` : graphical user interface clients.
    - `/sfm` : structure from motion client.
    - `/photo_capture` : photo capture client.
  - Clients are separated between them using a universal unique identifier as their id.
  - As a starting point, multiple graphical user interface clients are allowed, a single structure from motion, and a single photo capture.

Drawing 2: Coordinator's software modules



Drawing 3: Coordinator's event management

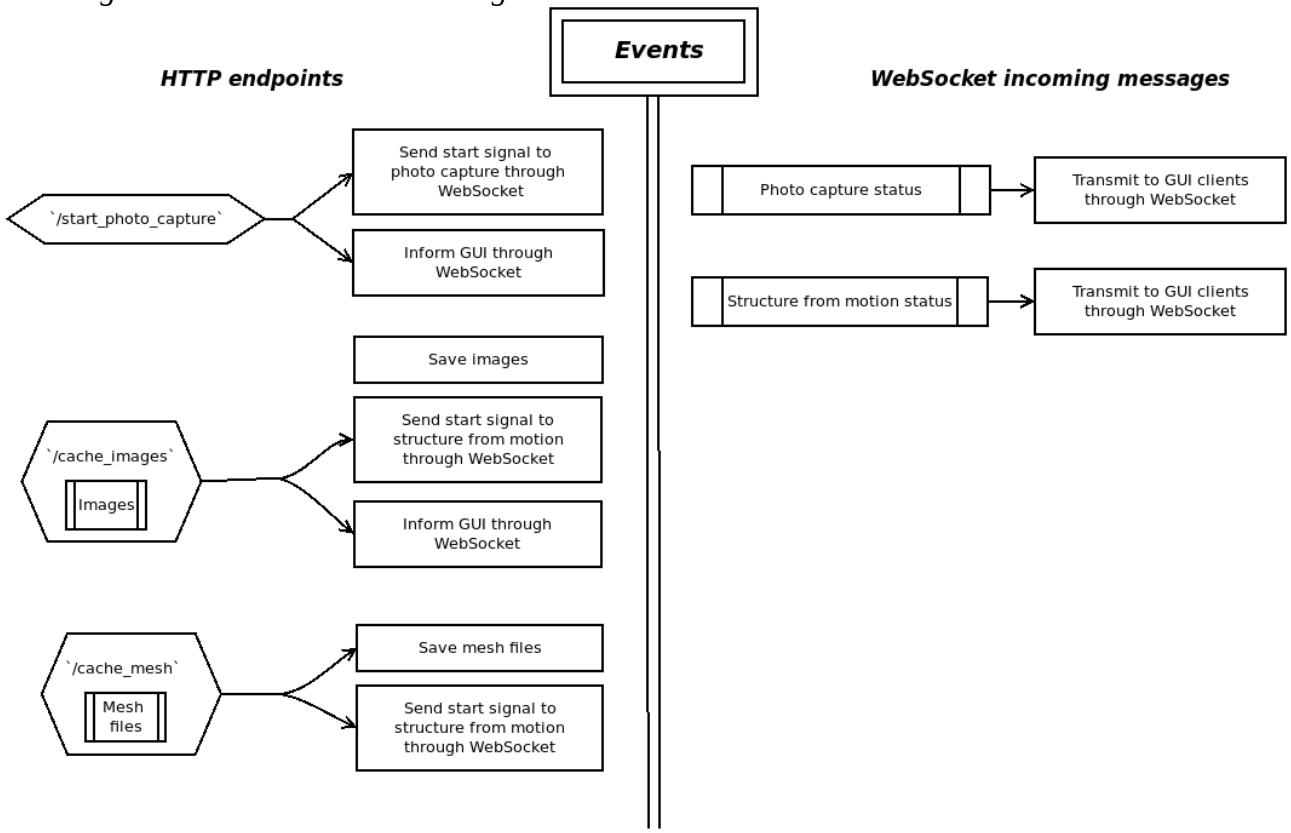


Photo capture

The path planning algorithm is the following:

- A semicircle is placed in object’s location, which is created by user input. This input is a) circle’s radius, b) object’s/circle’s position
- Some other obstacles are also placed accordingly (e.g. arm’s mount, table’s protecting sides)
- Path planning algorithm’s search space is reduced, not to include created obstacles.
- Discrete points are calculated.
- The path which connects the discrete points is calculated.
- Camera’s orientation for each point is calculated, it must always point in semicircle’s center.

The robotic arm is afterwards commanded to follow the route, take pictures on each point, and forward the images with necessary metadata for the next step, structure from motion.

[Mixalis Flowchart/block diagram/oti]

Structure from motion

Meshlab is configured to run the following pipeline:

1. Camera initialization.
2. Feature extraction.
3. Image matching.
4. Feature matching.
5. Structure from motion.
6. Dense scene preparation.
7. Depth map creation.
8. Depth map filtering.
9. Meshing.
10. Mesh filtering.
11. Texturing.

The main application is developed in Python:

- A WebSocket client establishes a connection with the coordinator and waits for messages. Once ‘start’ or ‘stop’ messages arrive, this module starts or stops its execution.
- An HTTP client wrapper downloads the images from the coordinator and posts mesh files to the latter.
- Meshlab is executed by spawning a new process.

Graphical user interface

A browser based application, developed in Javascript ES6. ReactJS is its main component, a component based, declarative framework for user interfaces.

- Buttons for starting/stopping photo capture and structure from motion.
- The completion status of the latter two is visualized using loading animations.
- Images are depicted via slide-show when they are uploaded by photo capture.
- Mesh is visualized using an appropriate 3d engine/wrapper.

Underlying, a WebSocket client is created and connected with coordinator's server.

1. When a user clicks a button, an HTTP get request is made on the appropriate endpoint. By pressing "start" photo capture, one can initiate the 3D reconstruction process as a whole. By pressing stop photo capture, a signal is sent to the specific module which instructs it to stop. Same as stop structure from motion. Start structure from motion button starts the procedure from 2nd step, considering that images (from photo capture) exist on the server already.
2. When photo capture sub process is completed, the coordinator sends a websocket message to the GUI informing the latter about images' paths. GUI gets those images (using HTTP GET request) and feeds them to the slide-show component. The slideshow appears in user's page.
3. When structure from motion sub process is completed, the coordinator sends another WebSocket message to the GUI informing the latter about mesh files' paths. The 3D library takes care of loading the image (GLTF format) and depicting it in a window. The user is able of rotating, zooming, moving the mesh inside that window.

Edge detection

Developments life-cycle

Stage 1: 3D Reconstruction

Complete 3D reconstruction.

1. Photo capture.
2. Structure from motion.
3. Coordination.
4. Graphical user interface.

Expected outcome

An object is placed in robotic arm’s space. The user inputs the object’s position and max dimension. The user clicks on “Start” photo capture button. The procedure should no longer expect user input, and complete by itself. Images, mesh, completion status is depicted.

Stage 2

Edge detection

Expected outcome

Step 3

Remote welding

Software licenses

Coordinator:

NodeJS	MIT-like: https://github.com/nodejs/node/blob/master/LICENSE	Internal package dependencies' licenses might differ
ws	MIT	
ExpressJS	MIT	
MaterialUI	Free for a free product: https://material-ui.com/store/license/	
Multer	MIT	
react-image-show	MIT	
cors	MIT	
obj2gltf	Apache-2.0	

Graphical user interface:

ReactJS	MIT	
ThreeJS	MIT	

Python interfaces

Requests	Apache-2.0	
websocket_client	BSD	

License templates

MIT template

- https://opensource.org/licenses/MIT

BSD template

- 3-clause: https://opensource.org/licenses/BSD-3-Clause
- 2-clause: https://opensource.org/licenses/BSD-2-Clause

Apache License, Version 2.0

- https://opensource.org/licenses/Apache-2.0