1 For Spatially Aware Decision Making Algorithms 1
2 Intro 3
3 Install 5
4 Motivating Examples 7
  4.1 mapServer package .......................... 7
  4.2 About ........................................ 16
5 Indices and tables 17
6 Credits 19
Python Module Index 21
For Spatially Aware Decision Making Algorithms
World Engine uses GDAL and Rasterio to interface with raw raster data in a variety of formats. A built-in socket server makes it possible to host this information via UDP to client applications. The intent of this project is to offer a framework for those wishing to implement decision making algorithms on top of geographic information.
World Engine is currently not configured or packaged for distribution. Stay tuned, as this functionality will be introduced soon:

```shell
pip install map_server
```
Motivating Examples

New in version version: Advanced API calls from Matlab/Simulink  Advanced request responder  Beta sphinx-autodoc

Changed in version version: None

Contents:

4.1 mapServer package

4.1.1 Subpackages

mapServer.mapping package

Submodules

mapServer.mapping.map_interface module

```
exception mapServer.mapping.map_interface.AddVehicleException(value)
   _bases: exceptions.Exception

If vehicle has invalid UUID (name) or does not exist
```

class mapServer.mapping.map_interface.MapInterface(filename)
   _bases: mapServer.mapping.world_engine.Map

Interface between Matlab and the MapData retrieval methods provided by ‘WorldEngine’ Used to interpret commands from UDP packets.

```
add_vehicle(vehicle)
    Add vehicle to the dictionary representing vehicles present on the map

Parameters vehicle –

init_position(xCoords, yCoords, vehicleName=None)
    Used to set the initial coordinates of the vehicle on the map. Can only be called once.

Parameters
    - xCoords – the x coordinate to be set
    - yCoords – the y coordinate to be set
```
• **vehicleName** – Name of the vehicle to be updated; if vehicle is not present, adds it with init_position. Of type UUID

**update_position**(xCoords, yCoords, vehicleName=None)

Update Position with the Map class. Gives new coordinates so that Map has the latest coordinates and elevation matrix. Should make a call to the Vicinity classes function for retrieving elevation matrix, update the elevation parameter of map_interface.

**Parameters**

- xCoords – the x coordinate to be set
- yCoords – the y coordinate to be set
- vehicleName – Name of the vehicle to be updated; if vehicle is not present, adds it with init_position. Of type UUID

mapServer.mapping.world_engine module

WorldEngine is used as the data layer for the threaded server. Retrieves and processes data stored in raster images. Manages

class mapServer.mapping.world_engine.Map(filename, verbose=False, **kwargs)

Bases: mapServer.mapping.world_engine.MapFile

The Map Class offers both atomic and advanced map read operations. The Map Class is built on top of the MapFile class for low-level file reading operations. Map depends on the GDAL and Rasterio abstraction layers.

**static distance_on_unit_sphere**(coord1, coord2, mode='km')

Calculates the distance on a unit sphere, with the radius of the earth hard-coded. Note that this formula is assuming a spherical earth.

**Parameters**

- coord1 – start coordinates
- coord2 – end coordinates
- mode – choose whether to return in km or feet

**Returns** return the distance in km

**get_coordinates_along_path**(segmentPairs, readMode='segments', **kwargs)

A path is a set of connected segments. Specifically, this function is to be called with an array of coordinates. Iterating through this array pairwise, we call the get_coordinates_in_segment() function iteratively forming a new, continuous array or coordinate connecting many segments.

**Parameters** segmentPairs – An array of coordinates representing the ‘vertices’ of the path to be traversed; depending on readMode, this can be either an array or a .csv file.

**Returns** A continuous set of coordinate along a path, which is a set of connected linear segments

**get_coordinates_in_segment**(startCoord, endCoord, mode='samples', numSamples=15, returnStyle='array')

Get coordinates along the direct path between start and end coordinates

**Parameters**

- startCoord – a Coordinate containing lat and lon, the starting point of the path.
- **endCoord** – a Coordinate containing lat and lon, the end point of the path.
- **mode** – determines how the elevation is sampled, either by pixel width, or a given sampling rate in

'coordinate distance':

- **numSamples**: Number of
- **returnStyle**: Default return style is an array of Coordinates

:returnStyle: not sure yet, perhaps determine this with an optional arg

### get_elevation_along_path(**kwargs)**
Query elevations along a path defined by a list of segments. Works by calling `get_elevation_by_segment()` iteratively on the segment list (CSV file)

**Parameters**
- **path** – Path specified as CSV file or coordinates

:return the distances between each coordinate, as well as the elevations at each coordinate

### get_elevation_along_segment(coordinateArray)
A segment is an array of coordinates, or similar iterable structure of coords. This function returns a conjugate array of elevations corresponding to each coordinate element in the input array

**Parameters**
- **coordinateArray** –

### get_point_elevation(**kwargs)**
Retrieve an elevation for a single Coordinate

**Parameters**
- **coordinate** – Named tuple of type Coordinate containing a lat/lon pair
- **mode** – Indicates whether we are passing in a Coordinate of lat/lon or a PixelPair of x/y

### get_surrounding_elevation(*args, **kwargs)**
Return a square matrix of size window x window

**Parameters**
- **mode** – Specify if we are going to read coordinates by lat/lon (‘coords’ mode) or by pixel in x/y (‘pixel’ mode)
- **window**: dimension of the square window to be read based on start Coordinates obtained
- **coordinates**: Named tuple of type Coordinate; cannot be specified if also specifying a vehicleName
- **vehicleName**: The UUID of a vehicle object; used to retrieve the vehicle’s current coordinates

**:return**: a square matrix with sides of length window

### latLonToPixel(coords)
First open the file with gdal (see if we can get around this), then retrieve its geotransform. Next, obtain a spatial reference, and perform a coordinate transformation.

Return the pixel pair corresponding to the input coordinates given in lat/lon

**Parameters**
- **coords**: A named Tuple of type ‘Coordinate’ containing a lat/lon pair

**:return**: A named tuple of type PixelPair containing an x/y pair

### static latlon_distance_on_unit_sphere(coord1, coord2, mode=’km’)
Compute either the lateral or longitudinal distance from one point to another; This corresponds to finding the length of one of the legs of the right triangle between the two points.

**Parameters**
- **coord1** – start coordinates
- **coord2** – end coordinates
- **mode** – choose whether to return in km or feet

**:Returns**
planPath(startCoord, endCoord, **kwargs)
   From start coordinates to end coordinates, sample elevation. Determine Path optional args will determine how the path is optimized

plot(**window)
   This function is not yet ready to call. Used to be executed in main after point and elevation samplings.

class mapServer.mapping.world_engine.MapFile(filename, verbose=False)
   MapFile abstracts the atomic raster file read/write processes.

exception mapServer.mapping.world_engine.ReadException(strn)
   ReadExceptions occur in response to invalid rasters or file paths
   Args:  strn (str): Human readable string describing the exception.
   Attributes: strn (str): Human readable string describing the exception.

exception mapServer.mapping.world_engine.RetrievePointException(strn)
   ReadExceptions occur in response to invalid queries on coordinates
   These invalid queries include coordinates that are not within the extent of the current raster, or those that are either not available or contain a recognized 'no-data' value.
   Note:  I'm unsure how NaN values translate on the Matlab side. It is also possible to return the 'no-value' float directly. This value can typically be found in a raster files header, and are stored in the MapFile class' attribute nodatavalue
   Args:  strn (str): Human readable string describing the exception.
   Attributes: strn (str): Human readable string describing the exception.

mapServer.mapping.world_engine.pairwise(iterable)
   s -> (s0,s1), (s2,s3), (s4, s5), ...

Module contents

mapServer.server package

Subpackages

mapServer.server.playground package

Submodules

mapServer.server.playground.maskingTape module

Module contents
Submodules

mapServer.server.matlab_socket module

class mapServer.server.matlab_socket.MyTCPHandler (request, client_address, server)
   Bases: SocketServer.BaseRequestHandler
   The RequestHandler class for our server.
   It is instantiated once per connection to the server, and must override the handle() method to implement communication to the client.
   handle()

class mapServer.server.matlab_socket.mapServer (host, port)
   Bases: object

mapServer.server.matlab_socket.openSocket (host, port)
   Open a socket on specified port with given host
   Parameters
      • host – host ID
      • port – port number to listen on
   Returns a socket object

mapServer.server.matlab_socket.serviceRequest (fileName)
   This function does something. :param fileName:

mapServer.server.server module

The server module is responsible for managing the threaded UDP socket server. This server is used to communicate with Matlab/Simulink simulations, but can also be used in the implementation of other decision making algorithms.

exception mapServer.server.server.CommandNotFound
   Bases: exceptions.Exception
   Command received does not match one listed in the command dictionary
   args
   message

class mapServer.server.server.Interrupt (interval, function, *args, **kwargs)
   Bases: object
   start()
   stop()

class mapServer.server.server.ThreadedUDPServer (server_address, RequestHandlerClass, bind_and_activate=True)
   Bases: SocketServer.ThreadingMixIn, SocketServer.UDPServer
   Threaded UDP server for receiving and responding to requests from client applications. We can run the server safely by doing:
   Example:
map_server = ThreadedUDPServer((HOST, PORT), UDP_Interrupt)
server_thread = None
logger.info('Instantiation succesful')
# terminate with Ctrl-C
try:
    server_thread = Thread(target=map_server.serve_forever)
    server_thread.daemon = False
    logger.info("Threaded server loop running in: {}".format(server_thread.name))
    print("Threaded server loop running in: {}".format(server_thread.name))
    server_thread.start()
except KeyboardInterrupt:
    server_thread.kill()
    map_server.shutdown()
    sys.exit(0)

address_family = 2
allow_reuse_address = False
close_request (request)
daemon_threads = False
fileno ()
    Return socket file number.
    Interface required by select().
finish_request (request, client_address)
    Finish one request by instantiating RequestHandlerClass.
get_request ()
    Override native get_request function in order to print out who is connecting to the server @todo make this Python3 compatible by using superclasses. Will need to update socketServer package :return:
handle_error (request, client_address)
    Handle an error gracefully. May be overridden.
    The default is to print a traceback and continue.
handle_request ()
    Handle one request, possibly blocking.
    Respects self.timeout.
handle_timeout ()
    Called if no new request arrives within self.timeout.
    Overridden by ForkingMixIn.
max_packet_size = 8192
process_request (request, client_address)
    Start a new thread to process the request.
process_request_thread (request, client_address)
    Same as in BaseServer but as a thread.
    In addition, exception handling is done here.
request_queue_size = 5
serve_forever \((\text{poll\_interval}=0.5)\)
Handle one request at a time until shutdown.
Polls for shutdown every poll\_interval seconds. Ignores self.timeout. If you need to do periodic tasks, do them in another thread.

server_activate()
server_bind()
Called by constructor to bind the socket.
May be overridden.

server_close()
Called to clean-up the server.
May be overridden.

shutdown()
Stops the serve_forever loop.
Blocks until the loop has finished. This must be called while serve_forever() is running in another thread, or it will deadlock.

shutdown_request(request)

socket_type = 2
timeout = None

verify_request(request, client_address)
Verify the request. May be overridden.
Return True if we should proceed with this request.

class mapServer.server.server.UDP_Interrupt(request, client_address, server)
Bases: SocketServer.BaseRequestHandler
This class works similar to the TCP handler class, except that self.request consists of a pair of data and client socket, and since there is no connection the client address must be given explicitly when sending data back via sendto().

command_response(cmd_name, returned_data, socket, client_ip, client_address)
Parse raw input and execute specified function with args

Parameters coords – coordinates in lat/lon

Returns the command and arguments as a dictionary

command_service(rawCommand)
Parse raw input and execute specified function with args

Parameters rawCommand – csv string from Matlab/Simulink of the form: ‘command, namedArg1, arg1, namedArg2, arg2, ..., namedArgN, argN’

Returns the command and arguments as a dictionary

finish()

static func_explode(s)

handle()
Handles UDP requests to the server. The map interface class is responsible for parsing the request, and executing the requested function.

Returns
setup()

Instantiate the connection with the worldEngine, the MapInterface. :rtype : None

static split_by_n(seq, n)

A generator to divide a sequence into chunks of n units.

mapServer.server.server.call_request(url=None, data=None, headers=None)

mapServer.server.server.grouper(iterable, n, fillvalue=None)

Collect data into fixed-length chunks or blocks grouper('ABCDEFG', 3, 'x') \rightarrow ABC DEF Gxx ;param iterable:

;param n: ;param fillvalue: ;rtype : object

mapServer.server.server_conf module

Configuration file for the socket server; contains information about location of database, Host, Port, and etc. Settings are contained in a dictionary named ‘settings’, currently holding the following information:

- Fileconfig
  - filename

- Host

- Port

Example:

An minimal settings dictionary:

settings = {

'FILE_CONFIG':
  {'filename':
    r'/Users/empire/Academics/UCSC/nasaResearch/californiaNed30m/elevation_NED30M_ca_2925289_01/
    virtRasterCalifornia.vrt'},

'HOST': 'localhost',

'PORT': 2002

}

Module contents

mapServer.utils package

Submodules

mapServer.utils.nmea_parser module

Credit to a1ronzo/gps_tracker

mapServer.utils.nmea_parser.position()
Module contents

mapServer.vehicles package

Submodules

mapServer.vehicles.vehicle module

The quadrotor class can be used to place various vehicles on a map. The class keeps track of the vehicles starting point, and its current location

```python
class mapServer.vehicles.vehicle.Coordinate(lat, lon)
    Bases: tuple
    lat
        Alias for field number 0
    lon
        Alias for field number 1

class mapServer.vehicles.vehicle.Path
    Bases: object

class mapServer.vehicles.vehicle.PixelPair(x, y)
    Bases: tuple
    x
        Alias for field number 0
    y
        Alias for field number 1

class mapServer.vehicles.vehicle.Quadrotor(initialLat, initialLon)
    Bases: mapServer.vehicles.vehicle.Vehicle

The Quadrotor Vehicle Class

This class is used as the base class for implementing particular quadrotor vehicles with specific weights, payloads, battery and thrust capacities.

    battery
    payload

    vehicle_type()
        "Return a string representing the type of vehicle this is.

    weight

class mapServer.vehicles.vehicle.Vehicle(initialLat, initialLon)
    Bases: object

    coordinates
    initialCoordinates
        Set the initial coordiantes of the vehicle

    range

    vehicle_type()
        "Return a string representing the type of vehicle this is.
```
Module contents

4.1.2 Module contents

4.2 About

This is a new page

4.2.1 Sub-heading 1

4.2.2 Sub-heading 2

Note: map_server is still in beta. Suggestions, contributions, or comments are welcome.

Warning: This software is still in development; as such, decision algorithms built on top of the module cannot be guaranteed.
Indices and tables

- genindex
- modindex
- search
CHAPTER 6

Credits

```bash
#.. include:: ../CONTRIBUTORS.txt
```
mapServer, 16
mapServer.mapping, 10
mapServer.mapping.map_interface, 7
mapServer.mapping.world_engine, 8
mapServer.server, 14
mapServer.server.matlab_socket, 11
mapServer.server.playground, 10
mapServer.server.playground.maskingTape, 10
mapServer.server.server, 11
mapServer.server.server_conf, 14
mapServer.utils, 15
mapServer.utils.nmea_parser, 14
mapServer.vehicles, 16
mapServer.vehicles.vehicle, 15
Index

A
add_vehicle() (mapServer.mapping.map_interface.MapInterface method), 7
address_family (mapServer.server.server.ThreadedUDPServer attribute), 12
AddVehicleException, 7
allow_reuse_address (mapServer.server.server.ThreadedUDPServer attribute), 12
args (mapServer.server.server.CommandNotFound attribute), 11

B
battery (mapServer.vehicles.vehicle.Quadrotor attribute), 15

C
call_request() (in module mapServer.server.server), 14
close_request() (mapServer.server.server.ThreadedUDPServer method), 12
command_response() (mapServer.server.server.UDP_Interrupt method), 13
command_service() (mapServer.server.server.UDP_Interrupt method), 13
CommandNotFound, 11
Coordinate (class in mapServer.vehicles.vehicle), 15
coordinates (mapServer.vehicles.vehicle.Vehicle attribute), 15

D
daemon_threads (mapServer.server.server.ThreadedUDPServer attribute), 12
distance_on_unit_sphere() (mapServer.mapping.world_engine.Map static method), 8

F
fileno() (mapServer.server.server.ThreadedUDPServer method), 12
finish() (mapServer.server.server.UDP_Interrupt method), 13
finish_request() (mapServer.server.server.ThreadedUDPServer method), 12
func_explode() (mapServer.server.server.UDP_Interrupt static method), 13

G
generate_coordinates_along_path() (mapServer.mapping.world_engine.Map method), 8
generate_coordinates_in_segment() (mapServer.mapping.world_engine.Map method), 8
generate_elevation_along_path() (mapServer.mapping.world_engine.Map method), 9
generate_elevation_along_segment() (mapServer.mapping.world_engine.Map method), 9
generate_point_elevation() (mapServer.mapping.world_engine.Map method), 9
grouper() (in module mapServer.server.server), 14

H
handle() (mapServer.server.matlab_socket.MyTCPHandler method), 11
handle() (mapServer.server.server.UDP_Interrupt method), 13
handle_error() (mapServer.server.server.ThreadedUDPServer method), 12
handle_request() (mapServer.server.server.ThreadedUDPServer method), 12
handle_timeout() (mapServer.server.server.ThreadedUDPServer method), 12

I
init_position() (mapServer.mapping.map_interface.MapInterface
initialCoordinates (mapServer.vehicles.vehicle.Vehicle attribute), 15

Interrupt (class in mapServer.server.server), 11

lat (mapServer.vehicles.vehicle.Coordinate attribute), 15
latlon_distance_on_unit_sphere() (mapServer.mapping.world_engine.Map static method), 9
latLonToPixel() (mapServer.mapping.world_engine.Map method), 9
lon (mapServer.vehicles.vehicle.Coordinate attribute), 15

Map (class in mapServer.mapping.world_engine), 8
MapFile (class in mapServer.mapping.world_engine), 10
MapInterface (class in mapServer.mapping.map_interface), 7
mapServer (class in mapServer.server.matlab_socket), 11
mapServer (module), 16
mapServer.mapping (module), 10
mapServer.mapping.map_interface (module), 7
mapServer.mapping.world_engine (module), 8
mapServer.server (module), 14
mapServer.server.matlab_socket (module), 11
mapServer.server.playground (module), 10
mapServer.server.playground.maskingTape (module), 10
mapServer.server.server (module), 11
mapServer.server.server_conf (module), 14
mapServer.utils (module), 15
mapServer.utils.nmea_parser (module), 14
mapServer.vehicles (module), 16
mapServer.vehicles.vehicle (module), 15
max_packet_size (mapServer.server.server.ThreadedUDPServer attribute), 12
message (mapServer.server.server.CommandNotFound attribute), 11
MyTCPHandler (class in mapServer.server.matlab_socket), 11

openSocket() (in module mapServer.server.matlab_socket), 11

pairwise() (in module mapServer.mapping.world_engine), 10
Path (class in mapServer.vehicles.vehicle), 15
payload (mapServer.vehicles.vehicle.Quadrotor attribute), 15
PixelPair (class in mapServer.vehicles.vehicle), 15
planPath() (mapServer.mapping.world_engine.Map method), 9
plot() (mapServer.mapping.world_engine.Map method), 10
position() (in module mapServer.utils.nmea_parser), 14
process_request() (mapServer.server.server.ThreadedUDPServer method), 12
process_request_thread() (mapServer.server.server.ThreadedUDPServer method), 12

Quadrotor (class in mapServer.vehicles.vehicle), 15

range (mapServer.vehicles.vehicle.Vehicle attribute), 15
ReadException, 10
request_queue_size (mapServer.server.server.ThreadedUDPServer attribute), 12
RetrievePointException, 10

serve_forever() (mapServer.server.server.ThreadedUDPServer method), 12
server_activate() (mapServer.server.server.ThreadedUDPServer method), 13
server_bind() (mapServer.server.server.ThreadedUDPServer method), 13
server_close() (mapServer.server.server.ThreadedUDPServer method), 13
serviceRequest() (in module mapServer.server.matlab_socket), 11
setup() (mapServer.server.server.ThreadedUDPServer method), 13
shutdown() (mapServer.server.server.ThreadedUDPServer method), 13
shutdown_request() (mapServer.server.server.ThreadedUDPServer method), 13
socket_type (mapServer.server.server.ThreadedUDPServer attribute), 13
split_by_n() (mapServer.server.server.ThreadedUDPServer static method), 14
start() (mapServer.server.server.Interrupt method), 11
stop() (mapServer.server.server.Interrupt method), 11

ThreadedUDPServer (class in mapServer.server.server), 11
timeout (mapServer.server.server.ThreadedUDPServer attribute), 13

UDP_Interrupt (class in mapServer.server.server), 13
update_position() (mapServer.mapping.map_interface.MapInterface method), 8
Vehicle (class in mapServer.vehicles.vehicle), 15
vehicle_type() (mapServer.vehicles.vehicle.Quadrotor method), 15
vehicle_type() (mapServer.vehicles.vehicle.Vehicle method), 15
verify_request() (mapServer.server.server.ThreadedUDPServer method), 13

weight (mapServer.vehicles.vehicle.Quadrotor attribute), 15

x (mapServer.vehicles.vehicle.PixelPair attribute), 15

y (mapServer.vehicles.vehicle.PixelPair attribute), 15