ASSIGNMENT 2 + QUIZ 2

MTRN2500 Computing for Mechatronic Engineers - S2 2018

Online Submission Due Date: 11:55pm, Sunday 16th September 2018
Quiz: Your laboratory class in Week 9
In-class Assessments: Your laboratory class in Weeks 7, 8 and 9
Weighting of final course mark: 20 % (Assignment) + 5 % (Quiz)

1 Learning Outcomes

This assignment specifically targets the following learning outcomes:

- 1. Be well versed with structured and modular programming using C/C++ and to appreciate the use of software to communicate with external devices.
- 3. Be able to develop prototype user interfaces to assist in the development of controlled Mechatronic systems.

2 Quiz (5 marks)

During your laboratory class in week 9 you will be given a written quiz immediately before the in-class assessment. This will include content from lectures in weeks 1-7, with an emphasis on the items used in this assignment. The quiz will begin at 10 minutes after the beginning of the class and you'll have 20 minutes to complete it. Normal exam conditions apply and you'll only need your student card and a pen. Feedback on this will be provided within 2 weeks of the quiz. Following the quiz, the in-class assessment (coding) for that week will begin.

3 Assignment Specification

This assignment is split into four components. A style component and three in-class assessments. This document will explain what you are required to do, the dates for each part, and the contributions each part will make to your overall assignment mark. Total marks available for this assignment are 20. You will complete this assignment in pairs.
The associated quiz has 5 marks available and will be completed individually.

This assignment will require you to extend the provided 3D world modelling software to model the motion of two or more ground vehicles. You will then be required to read data streams or an input device in real-time, use the data to control the graphical ground vehicles and display them in their environments.

- Style Component (5 marks)
  - This is due Sunday 16th September 2018 at 11:55pm by submission to Moodle (see below).
- In-class Assessment Part 1 (5 marks): Static Shapes and Vehicles
  - See Section 4 below. This will be demonstrated to your tutor during your week 7 tutorial.
- In-class Assessment Part 2 (5 marks): Moving Vehicles
  - See Section 5 below. This will be demonstrated to your tutor during your week 8 tutorial.
- In-class Assessment Part 3 (5 marks): Final Demo

– See Section 6 below. This will be demonstrated to your tutor during your week 9 tutorial, immediately following the quiz.

3.1 Style and version control component (5 marks)

For the style component you are required to submit all your .h and .cpp files together with the provided files. In addition, a PDF document showing evidence of the use of version control (e.g. exported commits from github) clearly identifying the authors must be included. All these files must be located in a folder named with your full zID (i.e. z1234567) then zipped and submitted to Moodle by 11:55pm on Sunday 16th August 2018.

A submission box in Moodle will be open for you to submit the .zip file. Feedback on this will be provided within 2 weeks of submission.

Things we will be looking for include:

- (1 mark) Consistent and neat structure
- (1 mark) Choice of names for variables and #defines
- (1 mark) Commenting and readability of the code
- (1 mark) Modularity: is the program broken up into well defined files, classes, functions?
- (1 mark) Evidence that version control (eg. git) has been used by both partners to contribute actively to the solution.

4 In-class Assessment Part 1 (5 marks): Static Shapes and Vehicles

4.1 Creating Basic 3D Shapes

Your first task is to extend the `Shape` class to implement a set of basic 3D shapes and eventually form a vehicle. When we say *extend* we mean the object oriented programming term - class derivation - rather than adding a large chunk of code to the `Shape` class itself. The `Shape` class contains position and orientation attributes, which are common to every 3D object. You need to implement classes at least for the following shapes outlined in this document. For simplicity, each object can be assigned one colour using three `floats` to specify the red, green and blue components of the colour. Look at the "`Shape.hpp`" file for more details on how to interface with and extend the `Shape` object.

4.1.1 Rectangular Prism

This shape should make use of three additional member attributes concerning the length of the shape in the three spatial dimensions (for example, `x_length`, `y_length` and `z_length`). You should also devise how the volume of the prism is positioned relative to it's internal x, y and z attributes and how the `rotation` variable rotates the object in the horizontal plane. One suggestion could be to define the (x, y, z) location as the center of the object and the object's rotation is applied about the origin of the object.

4.1.2 Triangular Prism

This shape should make use of additional member attributes to specify the dimensions of the triangular prism, the choice of which is left up to you. The length of the prism is an obvious choice for one of these attributes, but you should determine a method for specifying the shape of the triangular dimensions of the object. For example, two different approaches, among others, are to store:

- the three side lengths of the triangle, or

- two side lengths and an angle.

You should also remember to intelligently decide on a center for your object and how this relates to object rotation.

4.1.3 Trapezoidal Prism

You should decide on additional member attributes to specify the shape of this object. In particular you need to decide on a way of specifying the dimensions of the trapezium at the end of the prism in an efficient way.

4.1.4 Cylinder

You can use additional member attributes such as length or height along with radius to specify the dimensions of a cylinder. You may make use of the inbuilt "`gluCylinder`" or "`glutCylinder`" functions. A cylinder for this project must be a solid cylinder. That is, it should consist of a curved surface together with two circles on each end of the cylinder. The center of a cylinder might be defined as the point half way along the central axis of the cylinder for rotation purposes.

4.2 Vehicle Modelling

Using at least one of each of the basic 3D shapes we have defined in the previous section, you should implement your own custom vehicle. When implementing a model of a vehicle you should extend from the base "`Vehicle`" class provided with the initial collection of files. When constructing the model from basic shapes you should position shapes in the vehicle's local frame of reference. As a result, the "`draw`" function in your custom vehicle class should contain the following general structure:

```
void MyVehicle::draw()
{
// move to the vehicle's local frame of reference
glPushMatrix();
positionInGL();

// all the local drawing code

// move back to global frame of reference
glPopMatrix();
}
```

The base "`Vehicle`" class contains an "`update`" function which handles interpreting control input into vehicle motion using a basic mathematical model. Although it is not required, you may modify or improve this mathematical model in the "`Vehicle`" class. The "`update`" function, combined with the drawing structure above will position and orient the vehicle's model correctly in 3D space. You need to make sure you position shapes so that the vehicle is facing the positive x-axis in it's local frame of reference. For best results make your vehicles not longer than 4 units and not wider than 3 units.

4.3 Assessment for Part 1

Demonstrate your ability to draw these shapes before the end of your tutorial in week 7.

1. (1 mark) Create a RectangularPrism class extended from the Shape class and use it to draw a rectangular prism.

2. (1 mark) Create a TriangularPrism class extended from the Shape class and use it to draw a triangular prism.
3. (1 mark) Create a TrapezoidalPrism class extended from the Shape class and use it to draw a trapezoidal prism.
4. (1 mark) Create a Cylinder class extended from the Shape class and use it to draw a cylinder.
5. (1 mark) Create a custom MyVehicle class extended from the Vehicle class and use it to draw a static vehicle of your choice, making use of one of each of the shapes defined above.

5 In-class Assessment Part 2 (5 marks): Moving Vehicles

Your second task is to make a moving vehicle which has wheels and animate both the rolling and steering motions of those wheels. The rolling motion of the wheels must coincide with the motion of the vehicle and the model's motion should steer according to the wheels' steering angle. The vehicle path can be pre-programmed.

In addition, this part deals with dynamically adding vehicles into your program using a server, and making these remote vehicles move. In order to do this you will need to interface with a data source over an Ethernet/Wi-fi connection. We have provided a class (called `RemoteDataManager`) and a set of functions (declared in Messages.hpp) that hides away most of the low-level implementation detail for you. The `RemoteDataManager` class will connect over the internet to the UNSW robotics server `www.robotics.unsw.edu.au`. Communication with the server is bidirectional, and is used to synchronise your environment with the server's. To enable communication with the data server, uncomment the relevant line of code in the `idle()` function in the `main.cpp` file near this line:

```
//RemoteDataManager::Connect("www.robotics.unsw.edu.au","18081");
```

Most of the messages received from the data server will be handled by the provided code, with the exception of the "M"-vehicle model message. The "M" message contains one or more VehicleModel objects represented according to the data structures given in Table 1, where each VehicleModel has a vector of shapes. You will need to process each model object to instantiate the remote vehicles correctly, and in each model object, you will need to process the shape information. The relevant section of code can be found in the `main()` function in the file `main.cpp` near this line:

```
//otherVehicles[vm.remoteID] = new MyVehicle();
```

You should replace "MyVehicle" with the name of your custom vehicle class and uncomment the line of code. This will enable remote vehicles to be added to the map of other vehicles to be drawn and updated.

As can be seen, the information here is not any different to the information you used to draw the local vehicle. Hence, you can use the parts of software you used to draw the local vehicle to draw these remote vehicles in an identical manner provided you match up the data received from the data server to your own vehicle data representation. As soon as you have drawn your remote vehicles, they will begin to move according to the state data streaming from the data server. The part that moves the remote vehicles has already been completed for you and you therefore do not have to do this part. However, on connection you will need to tell the remote server what your vehicle looks like. The functionality to package and send the data has been provided by the `RemoteDataManager` networking software. You need to provide the code to fill in the outgoing data structure, given in Table 1. If the data you provide is beyond nominal range, the server may respond with an error message and terminate your connection.

5.1 Assessment for Part 2

Demonstrate your ability to draw moving vehicles before the end of your tutorial in week 8.

Table 1: "M" Message

```
enum ShapeType
{
UNKNOWN_SHAPE,
RECTANGULAR_PRISM,
TRIANGULAR_PRISM,
TRAPEZOIDAL_PRISM,
CYLINDER
};

union ShapeParameter
{
struct RectangularParameters
{
float xlen;   // length along x-axis
float ylen;   // length along y-axis
float zlen;   // length along z-axis
} rect;

struct TriangularParameters
{
float alen;   // length of side A (bottom)
float blen;   // length of side B (left)
float angle;  // angle (degrees) between side A and B
float depth;  // length along z-axis
} tri;

struct TrapezoidalParameters
{
float alen;   // length of side A (bottom)
float blen;   // length of side B (top)
float height; // distance between side A and B
float aoff;   // distance A is shifted from B by, from the left
float depth;  // length along z-axis
} trap;

struct CylinderParameters
{
float radius;
float depth;  // length along z-axis

bool isRolling; // needs to roll with vehicle?
bool isSteering;// needs to steer with vehicle?
} cyl;
};

struct ShapeInit
{
ShapeType type;
ShapeParameter params;
float xyz[3];
float rotation;
float rgb[3];
};

struct VehicleModel
{
int remoteID;
std::vector<ShapeInit> shapes;
};
```

Table 2: "S" (State) Message

```
struct VehicleState
{
int remoteID; // this should be 0 for local vehicles
float x;
float z;
float rotation;
float speed;
float steering;
};
```

- (1 mark) Instantiate local/remote vehicles extended from Vehicle class.
- (1 mark) Vehicles should have wheels that roll when driving forward/backward.
- (1 mark) Vehicles should have front wheels that steer when steering.
- (1 mark) Implement code in your graphical application to receive data server message "M" and display one or more vehicles accordingly with the correct shapes.
- (1 mark) Implement code in your graphical application to report your local vehicle status to the server.

6 In-class Assessment Part 3 (5 marks): Final Demo

This part requires preparation before you come to the assessment. Learn to draw vehicles of arbitrary shape from a diagram. As good preparation, you could implement some prefabricated vehicle parts to make constructing vehicle models quicker. For example, a "`Wheel`" class could be created that contains a cylinder wheel with either cylindrical or prism based spokes for the wheels to tell they are turning when the vehicle is in motion.

Vehicle speeds will range from 0 to 10 ms^{-1} and steering angles will range between -15^o and $+15^o$ degrees.

6.1 Assessment for Part 3

During the week 9 tutorial, you will be asked to draw a vehicle of specified dimensions, given in the format as shown in Appendix A. You will be given 30 minutes to implement this vehicle using only the lab computers. You will have incorporated all other functionality required before arriving at the assessment. As such when you connect to the data server, we should see the remote vehicles on the screen and they should begin to move automatically. After such time your demonstrator will check the model you created to see if it appears to be correct to specification and any moving parts properly animate under vehicle motion. Your demonstrator will also check the reporting of your local vehicle status to the data server. You will then be asked to demonstrate your program by driving your vehicle around on the screen using an XBox controller.

Marking criteria

- (1 mark) Making the program compile and be fully operational on the day of assessment.
- (1 mark) Whether the model looks and animates correctly.
- (1 mark) The vehicle motion is demonstrated using an XBox 360 game controller.
- (1 mark) Whether the live feed of data from the data server is correctly read and used.
- (1 mark) Whether the local vehicle's status is correctly reported to the data server.
- If you are up for a challenge, demonstrate to your tutor how you could give chase to vehicle with vehicle ID 1, when you press 'L'. You will get an additional 1 course mark if successful.

7 Additional Information

- The assignment will be completed in pairs, both coding and demonstration and marks will be award to pairs. Select your partner on the Moodle group formation activity.
- A plagiarism check will be performed on all assignments and any instances of plagiarism will be dealt with under the UNSW plagiarism policy (linked from the course outline).
- Late submissions are not accepted, as per the course outline, unless Special Considerations has been granted. Refer to the course outline for how to do this if you are unsure.
- If you miss your in-class assessment or quiz, you are not permitted to join another class and must apply for Special Considerations to sit another.
- Please post questions about this assignment and quiz on the class Moodle discussion forum.
- Finally, enjoy learning how to work with OpenGL.

8 Getting Started

8.1 Setting Up

Setup instructions for Windows are provided in the Week 6 tutorial on Moodle. `https://moodle.telt.unsw.edu.au/pluginfile.php/3614572/mod_resource/content/2/wk6tut.html`

Setup instructions for Linux and Mac are available in Appendix B.

Although there are different steps for each operating system, all of the base assignment code can be found in the **AssignmentGL-Base.zip** file on Moodle. All supporting headers, libs and dlls can be found in the **AssignmentGL-Support.zip** file on Moodle.

8.2 Supplied Code Overview

You should begin this assignment by downloading the base code ZIP file from Moodle. A number of files have been provided to allow you to view and navigate a simple 3D world so you can test your code. See Section 8.1 for details on downloading and setting up the given base code.

Once the downloaded code is up and running the list of controls in Table 3 will allow you control the vehicle and to move the virtual camera.

Table 3: Base code commands

Control	**Description**
Arrow keys	Drive vehicle forward/left/backwards/right.
W,A,S,D	Move camera forward/left/backward/right.
C	Descend camera vertically.
(space)	Ascend camera vertically.
Mouse drag	Rotate the camera's viewing direction.
0 (zero)	Move the camera to the origin.
P	Move the camera to vehicle pursuit position.

Additionally, you can exit the program by pressing Escape. See the use of the "`KeyManager`" class in the main source file for more information on how keyboard events are linked to the virtual OpenGL Camera. In short, keys that are pressed once (such as getting the virtual camera to move to the origin) are handled the normal way via GLUT (Open**GL** **U**tility **T**oolkit). To handle multiple keys being held down at the same time,

the additional functionality of the custom "KeyManager" class is used.

Usually when modelling a vehicle's pose and motion from live sensor data you would acquire the data from a suite of on board sensors, a centralised database or a network socket. For this assignment, you will be reading data from a data server.

The "Shape" class provided should be used as the parent or base class for all shapes outlined in Section 4.1. For simplicity, all shapes you create will have a 3D position using (x, y, z) coordinates and a yaw angle θ in degrees. Yaw is the rotation in the horizontal plane, which in this case is the XZ-plane. Zero degrees of rotation means you are facing a direction parallel to the positive x-axis. The vertical axis is the y-axis and we are using a right hand coordinate system.

The "Vehicle" class should be used as the parent or base class for designing vehicles as outlined in Section 4.2. The base vehicle object uses a basic mathematical model to translate speed and steering values from input devices such as the keyboard or an Xbox controller to motion. Your initial task is to practice writing derived classes that represent different vehicle designs. See Section 4.2 for further details.

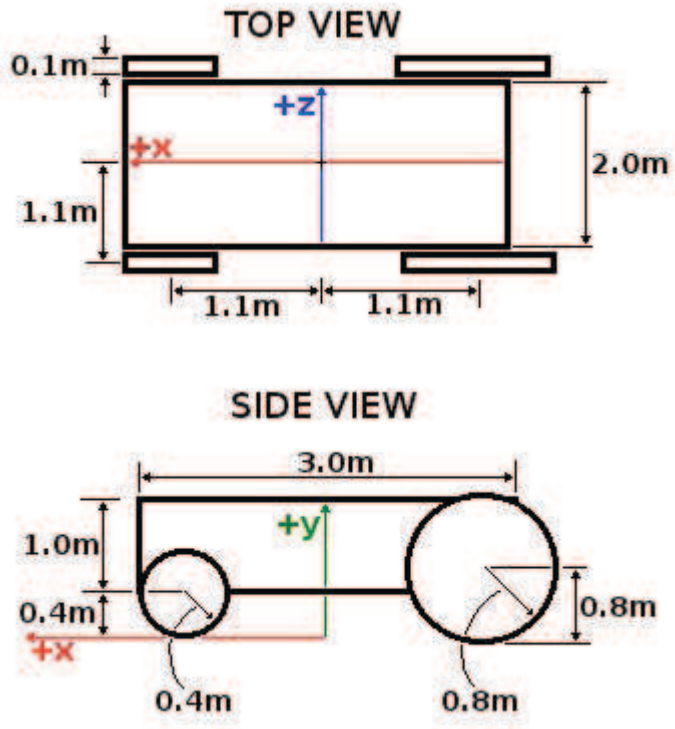
After you have defined a custom class deriving from the `Vehicle` class you can test it by finding the following section of code in the `main()` function in the file `main.cpp`:

```
// vehicle = new MyVehicle();
```

You should replace "MyVehicle" with the name of your custom vehicle class and uncomment the line of code. This will enable the rest of the keyboard and drawing functions to be linked with your custom vehicle class.

8.3 Appendix A: An Example Vehicle to Model

Below is a sample specification for a simple vehicle.



- The x-axis, y-axis and z-axis are shown by the red, green and blue vectors respectively.
- The small front wheels should be turned to reflect the vehicle's steering angle.
- All four wheels should rotate relative to vehicle speed.
- The main body should be coloured with the RGB values (0.0, 0.6, 0.0).
- The front (small) wheels should be coloured with the RGB values (1.0, 0.0, 0.0).
- The back wheels should be coloured with the RGB values (0.0, 0.0, 1.0).

8.4 Appendix B: setup for Linux and Max

These steps are based on using the g++ compiler.

Linux and Mac OS X are relatively the same, except on most Linux distros the compiler and libraries are installed by default. In Linux, if you try to run the command g++ and the shell complains that it cannot find the command, you should install the compiler by running the following command:

```
Ubuntu/Debian:
apt-get install build-essential
RedHat:
rpm install build-essential
```

On Mac OS X you will need to install Xcode. Xcode is available via the App Store for free, but a link can also be found on `http://developer.apple.com` if you have a free Apple developer licence. Xcode is something like 3 or 4 GB, so be warned.

8.4.1 Setting up a folder structure

1. Create a new directory called, for example, `assign2`.
2. Download the **AssignmentGL-Base.zip** file and extract the `hpp` and `cpp` files to the newly created directory.

8.4.2 Setting up OpenGL headers and libs

On Mac OS X, downloading Xcode will automatically set up OpenGL, GLU and GLUT in the correct place. On Linux you should be able to `apt-get install` (Debian/Ubuntu) or `rpm install` (RedHat) the correct packages. If on Linux and you can't install the headers and libs using a packing service, use the following backup steps:

1. Download the **AssignmentGL-Support.zip** file from Moodle and extract the files to an easy to find temporary location. In the Linux folder there should be two folders: `include` and `lib`.
2. Copy the `include` folder to a location that's easy to find (for example: `~/include`).
3. Copy the `lib` folder to a location that's easy to find (for example: `~/lib`).

Please ask for assistance in the Assignment Two **Discussion Board** on Moodle if you run into problems.

8.4.3 Compiling and linking with OpenGL

It is recommended to put the following text in a `Makefile` file in the assignment code directory.

```
LIBS = -lGL -lGLU -lGLUT
SRC =  <list all your cpp files>
LIBDIR = -L~/lib
INCDIR = -I~/include
all:
<tab> g++ -o run $(SRC) $(LIBS) $(LIBDIR) $(INCDIR) -g
```

On Mac OS X you may not need to specify the `LIBDIR` or `INCDIR` for OpenGL support files as Xcode might have set them up properly for you. Also, Mac OS X users might find it easier developing directly in Xcode instead of using g++ directly. There are plenty of tutorials online for setting up an OpenGL/GLUT project

with Xcode, but if you need further assistance, please ask in the discussion forum on Moodle.

By creating a `Makefile` you can simply type `make` at the command line in the assignment directory and it will compile and link your code to from an executable. Remember, when you add more cpp files make sure you make the necessary changes to the `Makefile` as well.

You can then run the program by typing:

```
./run
```