Mars-Lander

Please read through this documentation and setup the contants correctly when trying to test my autopilot. PLEASE DO REMEMBER to restart the program when you try to switch to another scenario just in case of any unexpected error

Features

Acceleration

Using the following function to add drag force/wind flow/gusts to pure acceleration produced by gravitational force

```
vector3d getRelativeVelocity(vector3d absolute_velocity);
```

Using the following function to obtain relative velocity (useful when trying to involve wind flow/gusts)

```
vector3d getRelativeVelocity(vector3d absolute_velocity);
```

Wind Flow

Wind Flow Control Constant (lander.h)

```
// Wind Flow Constant
#define WIND_FLOW_SPEED 1 // wind flow speed on ground (m/s)
// wind flow direction, two conponents of the direction vector
// eg: (1, 1) points to north east.
#define WIND_FLOW_DIRECTION_NORTH 1 //negative number indicates south
#define WIND_FLOW_DIRECTION_EAST 1 //negative number indicates west
#define WIND_FLOW_RANDOM_CONSTANT 1 //speed of additional random gusts (m/s)
```

Switch Integration Mode

lander.h

```
#define MODE_VERLET 1
#define MODE_EULER 0
```

lander.cpp:

```
bool integration_mode = MODE_VERLET;
```

Basic Autopilot Concept

Two-dimensional automatic control system

Sometimes the aircraft needs to be speed up or slowed down to a specific velocity. However, it's not efficient at all to adjust the velocity in different dimensions one by one. So we need a two-dimensional automatic control system to adjust the velocity both vertically and horizontally at the same time.

Delta Setting

When the error is small, the engine should provide a thrust which can cancel the effect of gravitational force. (Only apply in vertical velocity control)

```
//setup delta value
delta = gravitational_force / MAX_THRUST;
```

Throttle Setting and Attitude Change

The following code enables us to change speed in two-dimension

```
//set thrust
if (delta_r > 1 || delta_r < 0){
    cout<<"Delta value error: "<<delta_r<<endl;
    return;
}
if (Pout_r<=-delta_r){
    if (Pout_r <= -delta_r - 1){
        throttle1 = -1;
    }else{
        throttle1 = (Pout_r + delta_r);
    }
}
else if (Pout_r >= (1-delta_r)){
    throttle1 = 1;
}else{
    throttle1 = Pout_r + delta_r;
}

if (Pout_t >= 1){
    throttle2 = 1;
}else if (Pout_t<=0.1){
    throttle2 = 0;
}else{
    throttle2 = Pout_t;
}

vector3d new_attitude = v_t.norm() * throttle2  + e_r * throttle1;
attitude_autochange(new_attitude);
double new_throttle = sqrt(pow(throttle1,2) + pow(throttle2,2));
```

```
if (new_throttle >= 1) {
    throttle = 1;
}else{
    throttle = new_throttle;
}
```

Bouncing

Sometimes the aircraft will bounce around a specific velocity due to the overshoot of engine. The solution is to setup Kh and Kp properly. Lower down Kp can significantly eliminate the bouncing effect.

Error Reference Setting

Currently according to the knowledge given by handout, we are using a linear reference for our control system:

```
error = -(TARGET_VELOCITY + Kh * h + e_r * getRelativeVelocity(velocity));
Pout = Kp * error;
```

It's possible to apply a non-linear reference for the velocity. For example:

```
reference_velocity = Kh * pow((Height - Target_Height), 2)
```

It's possible to change the Kh by codes, which means the Kh value can fit in different scenarios. For example, when injecting my lander into orbit from below the atmosphere, I implemented this:

```
// initialize
Kh_r = (v_r.abs() / EXOSPHERE);
start_v_t = sqrt(GRAVITY * MARS_MASS*(1 / (MARS_RADIUS + EXOSPHERE) - 1 / perigee)
* 2 / (1 - pow(MARS_RADIUS + EXOSPHERE,2)/pow(perigee,2)));
cout<<"set start v_t "<<start_v_t<<endl;
```

Which initialize the Kh value dynamically and this injection function can fit into different injection tasks such as Scenario 7 (will be explained later).

Pilot Period

It's definitely not efficient to activate engine all the time. Therefore I designed an autopilot queue, using which we can divide the whole task into a sequence of tasks. For example, in Scenario 7, when trying to inject the lander which is initailly in the air and with no speed, we need to speed up the lander, and then excute the injection.

```
if (action_mode==MODE_1_PRE_INJECTION){
    double perigee = injection_orbit_perigee;
```

```
    double to_v_t = sqrt(GRAVITY * MARS_MASS*(1 / (MARS_RADIUS + EXOSPHERE) - 1 /
perigee) * 2 / (1 - pow(MARS_RADIUS + EXOSPHERE,2)/pow(perigee,2)))/2;
    autopilot_orbital_pre_injection((EXOSPHERE+(position.abs()-MARS_RADIUS))/2,
to_v_t, 2000);

}else if(action_mode==MODE_1_INJECTION){
    autopilot_orbital_injection();
}
```

For Scenario 3, the lander has an initial velocity, so we can excute injection directly.

```
autopilot_orbital_injection();
```

Scenarios

Scenario 1

Please make sure:

```
#define FUEL_RATE_AT_MAX_THRUST 0.5 // (l/s)
```

With the autopilot

The initial absolute velocity of the lander is zero but the ground speed is not due to the self rotation of the Mars. So we need a two-dimensional automatic control system to perform the landing The program analyzes the power needed to slow down the aircraft in two-dimension, and then automatically changes the attitude of the lander and launches the engine. (This is the application of my core autopilot concept)

Scenario 3

Please make sure:

```
#define FUEL_RATE_AT_MAX_THRUST 0.0 // (l/s)
```

Without auto pilot

The lander can not resist the drag force and finally crashes.

With auto pilot

```
// settings for orbital injection
// apogee > perigee
double injection_orbit_apogee = (MARS_RADIUS + 17032000) * 1.3;
double injection_orbit_perigee = MARS_RADIUS + 17032000;
```

```
**This is a very automatic system, which helps you inject the aircraft into almost "any" circular/ellipse orbit!**
```

The lander will firstly reach the top of atmosphere (where drag doesn't affect our movement too much), and then speed up until a certain velocity. With this velocity, lander resists the gravitational force and reach the perigee. Once it reaches perigee, engine is activated again and speed up to higher velocity so that it can move between apogee and perigee we provided.

Scenario 5

Please make sure:

```
#define FUEL_RATE_AT_MAX_THRUST 0.0 // (l/s)
```

This is similar to Scenario 7, using completely the same code.

Scenario 6

Please make sure:

```
#define FUEL_RATE_AT_MAX_THRUST 0.5 // (l/s)
```

Without auto pilot

The lander runs on an aerostationary orbit.

With auto pilot

The lander starts to slow down by activating the engine and pushing in opposite direction of velocity. After slowing down a specific speed, the engine is deactivated. The lander will follow an eclipse orbit to approach the atmosphere by the effect of gravitational force. Then the engine is started again, slow down the lander and release the parachute. Safely landed.

Scenario 7

This scenario has the same initial settings with **Scenario 1** but has an autopilot to inject the lander to an orbit. Please make sure:

```
#define FUEL_RATE_AT_MAX_THRUST 0.0 // (l/s)
```

The lander will firstly excutes a pre-injection period, in which it will speed up and fly higher. Then it will excute the same approach in Scenario 3 and insert into the desired orbit.

Tuning Autopilot

How to save fuel

The concept is to increase the effiency of the thrust

Solution 1: Make use of drag force

The drag force increases with speed, which means if we pass through the atmosphere with a higher speed, the drag force will produce more work to reduce the potential energy as well as the kinetic energy. Therefore, by tuning the Kh (likely to be lower), we can postpone the launch of engine and therefore provide a higher speed when moving downwards.

Solution 2: Make use of wind flow

The drag force depends on the relative speed regarding the atmosphere while not the absolute speed in 3D space. Therefore when trying to land, if posible we can fly from east to west and when trying to launch we'd better launch towards east to reduce the effect of drag force.

Solution 3: Make use of parachute

The drag force increases suddenly when we deploy the parachute. So slowing down the lander quickly and releasing the parachute (by turning Kh up in most cases) is also a solution.

Solution 4: Make use of gravitational force

I use this concept almost in every scenario, which helps the lander start approaching the surface of the planet with a calculated velocity. Also, it works when you are trying to inject the lander to an orbit. Below is the codes to calculate the velocity we need to approach the lander to the Mars surface.

```
a  = (position.abs() + MARS_RADIUS) / 2.0;
c = a - MARS_RADIUS;
launch_velocity = sqrt((a-c)* GRAVITY * MARS_MASS / (a*(a+c)));
```

Conclusion:

These solutions sometimes seem to be contradictory, so striking a balance is a tricky work!

Minimize descent time

There is no doubt that we need to start the engine as late as possible. But what the engine can do is limited, we need to take care of Kh&Kp setting (sometimes turning up Kp is useful) in case that the lander touches the ground too hard even with full throttle.

Minimize peek acceleration

This is basically contradictory to saving fuels. To minimize the peek acceleration, we need a more smooth descent. By setting:

```
Kh = (Speed_When_Start_Descending) / (Total_Descent_height)
```

I can get a very smooth descent, with a almost linear velocity change. But in practice, linear descent takes too much time. It's possible to apply a non-linear reference for the velocity. For example, giving a non-linear reference:

```
reference_velocity = Kh * function(Height - Target_Height)
```

The function should be a curve with an **as small as possible** first derivative. (quadratic function successfully reduce the peek acceleration in my tests.)

Engine Lag & Delay

Kp is essential to deal with engine lag & delay. According to my experiences, the major problem of engine lag & delay is bouncing. The engine reacts too late to error changes, which will sometimes cause the lander bouncing near the ground. It's very practical to use a lower Kp, which means the engine will not overshoot the error changes. Some successful trials:

```
#define ENGINE_LAG 0.5 // (s)
#define ENGINE_DELAY 0.5 // (s)
```

In Scenario 1, reduce Kp from 0.5 to 0.2. In Scenario 6, reduce Kp from 0.5 to 0.4.

Copyright