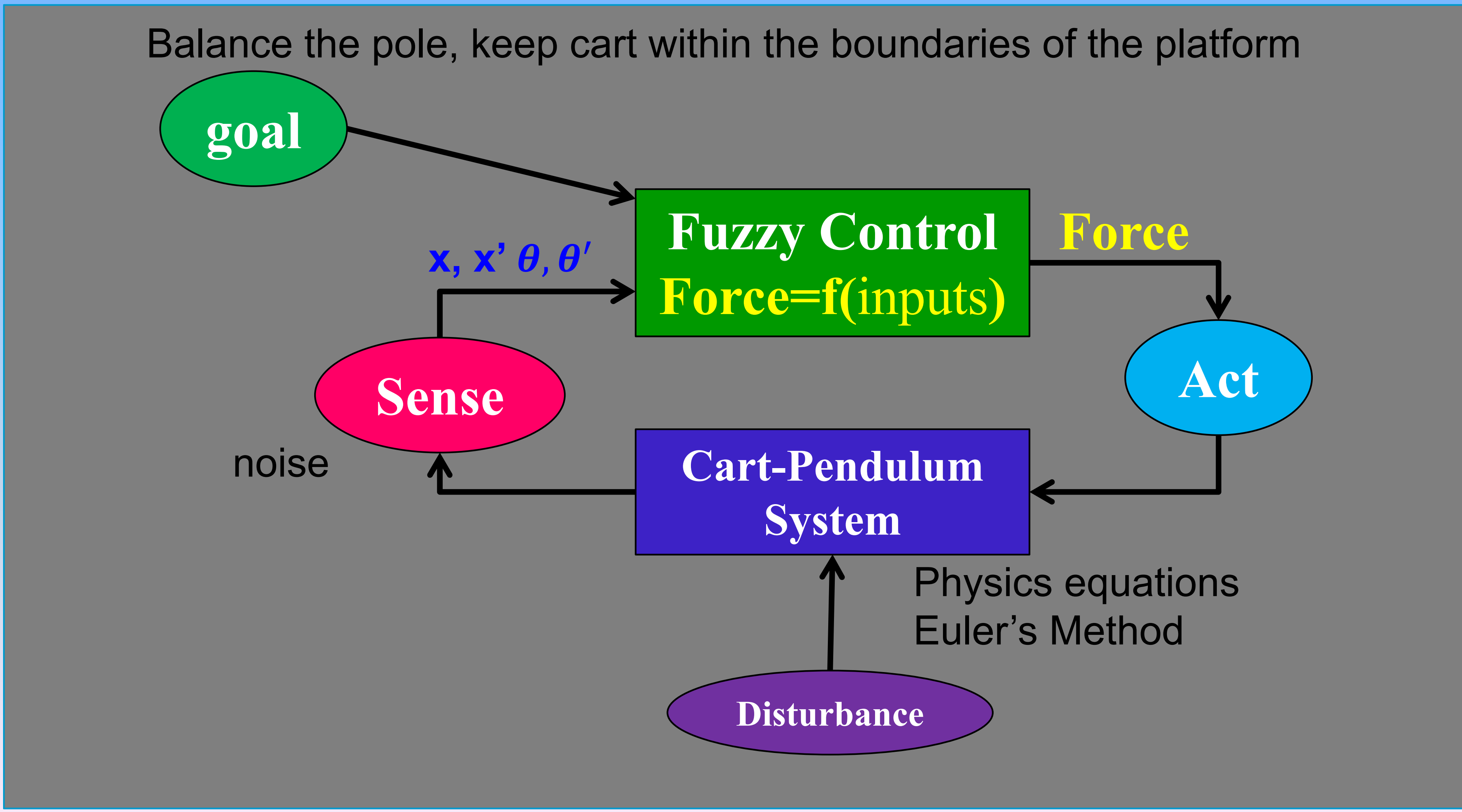
Assignment #2

Yamakawa’s Fuzzy Controller

- INSTRUCTIONS
- DOCUMENTATION GUIDE
- MEMBERSHIP FUNCTION VIEWER
- CHECKLIST

Feedback Control/Closed Loop Control

- Closed-loop control allows for **uncertainty** in the model as well as **noise** and **disturbances** in the system under control



Solve the tutorial first, before studying this lecture.

Fuzzy Logic Tutorial

2 Inputs, 9 rules

Fuzzy Controller

Start-up codes

SEARCH

Cross-platform Start-up Code: makefile

The start-up system is comprised of multiple files.

Executable filename

Windows support

macOS support

Linux support

```
makefile

CC := g++
TARGET := main

# Detect the operating system
ifeq ($(OS),Windows_NT)

	CFLAGS := -O2 -std=c++14 -Wall -c -fpermissive -fconserve-space -Wno-write-strings
	LFLAGS := -lgdi32

	EXTENSION := .exe
	CLEANUP := del
	CLEANUP_OBJS := del *.o

	# Find all source files (.cpp) and header files (.h)
	SRCS := main.cpp graphics.cpp transform.cpp sprites.cpp fuzzylogic.cpp
	HDRS := graphics.h transform.h sprites.h fuzzylogic.h
else
	UNAME_S := $(shell uname -s)
	ifeq ($(UNAME_S),Darwin)
		# macOS
		EXTENSION := .out
		CFLAGS := -O2 -std=c++14 -Wall -I/usr/local/include -L/usr/local/lib -c -Wno-write-strings
		LFLAGS := -L/usr/local/lib -lSDL_bgi -lSDL2
		CLEANUP := rm -f
		CLEANUP_OBJS := rm -f *.o

		# Find all source files (.cpp) and header files (.h)
		SRCS := main.cpp transform.cpp sprites.cpp fuzzylogic.cpp
		HDRS := transform.h sprites.h fuzzylogic.h

	else ifeq ($(UNAME_S),Linux)
		# Linux
		EXTENSION := .out
		CFLAGS := -O2 -std=c++14 -Wall -I/usr/local/include -L/usr/local/lib -c -Wno-write-strings
		LFLAGS := -lSDL_bgi -lSDL2
		CLEANUP := rm -f
		CLEANUP_OBJS := rm -f *.o

		# Find all source files (.cpp) and header files (.h)
		SRCS := main.cpp transform.cpp sprites.cpp fuzzylogic.cpp
		HDRS := transform.h sprites.h fuzzylogic.h
	endif
endif
```

Indented statements are preceded by a **tab**.

The program requires that you "**build**" it using a **makefile**.

SEARCH

Cross-platform Start-up Code: makefile

makefile

Windows support

macOS support

Linux support

```
CC := g++
TARGET := main

# Detect the operating system
ifeq ($(OS),Windows_NT)

    CFLAGS := -O2 -std=c++14 -Wall -c -fpermissive -fconserve-space -Wno-write-strings
    LFLAGS := -lgdi32

    EXTENSION := .exe
    CLEANUP := del
    CLEANUP_OBJS := del *.o

    # Find all source files (.cpp) and header files (.h)
    SRCS := main.cpp graphics.cpp transform.cpp sprites.cpp fuzzylogic.cpp
    HDRS := graphics.h transform.h sprites.h fuzzylogic.h
else
    UNAME_S := $(shell uname -s)
    ifeq ($(UNAME_S),Darwin)
        # macOS
        EXTENSION := .out
        CFLAGS := -O2 -std=c++14 -Wall -I/usr/local/include -L/usr/local/lib -c -Wno-write-strings
        LFLAGS := -L/usr/local/lib -lSDL_bgi -lSDL2
        CLEANUP := rm -f
        CLEANUP_OBJS := rm -f *.o

        # Find all source files (.cpp) and header files (.h)
        SRCS := main.cpp transform.cpp sprites.cpp fuzzylogic.cpp
        HDRS := transform.h sprites.h fuzzylogic.h

    else ifeq ($(UNAME_S),Linux)
        # Linux
        EXTENSION := .out
        CFLAGS := -O2 -std=c++14 -Wall -I/usr/local/include -L/usr/local/lib -c -Wno-write-strings
        LFLAGS := -lSDL_bgi -lSDL2
        CLEANUP := rm -f
        CLEANUP_OBJS := rm -f *.o

        # Find all source files (.cpp) and header files (.h)
        SRCS := main.cpp transform.cpp sprites.cpp fuzzylogic.cpp
        HDRS := transform.h sprites.h fuzzylogic.h
    endif
endif
```

SDL2 and **SDL_bgi** libraries require separate installation steps for macOS and Linux systems

Fuzzy Logic Engine

C/C++ version

Neural Network and Fuzzy Logic Applications in C/C++ (Wiley Professional Computing) by Stephen Welstead

Fuzzy Inference System

float inputs[MAX_NO_OF_INPUTS]

fuzzy_system_rec fz

crisp_output = fuzzy_system(inputs, fz);

A function that can execute a complete fuzzy inference system.

We would like a function that feeds on an array of inputs and a complete specification of a fuzzy system, and in turn returns the final crisp output.

Linguistic Terms

Enumerated data types – for easy indexing

fuzzylogic.h

// **Indices**

//

typedef **enum** {in_angle, in_distance}; //input indices

//indices of fuzzy sets for **angle input**

typedef **enum** {in_negatively_small, in_negatively_medium, in_negatively_large, in_zero,
in_positively_small, in_positively_medium, in_

//indices of fuzzy sets for **distance input**

typedef **enum** {in_near, in_medium, in_f

//indices for output **steering angle**

typedef **enum** {out_ze_turn, out_negativ
out_negatively_sharp_turn, out_negativel
out_positively_very_mild_turn, out_positi
out_positively_very_sharp_turn
};

Define **enum** variables for the linguistic terms.

We would like to refer to the different fuzzy sets and input parameters using descriptive names (e.g. {in_angle, in_position}, {in_negatively_small, in_negatively_large, etc.}), instead of numbers whenever we use them.

//indices for **output speed**

typedef **enum** {out_very_slow, out_slow, out_medium, out_fast, out_very_fast,
out_wicked_fast};

Add linguistic terms tailored to the problem domain

Trapezoidal Membership Function

fuzzylogic.h

```
//Trapezoidal membership function types
typedef   enum { regular_trapezoid, left_trapezoid, right_trapezoid }  trapz_type;


// Trapezoidal Fuzzy Set ////////////////////

struct  trapezoid {

  trapz_type    tp;  //type of trapezo

  float  a, b, c, d, l_slope, r_slope; /

};
```

Define a structure for holding the properties of the membership functions to use.

Here we have a structure implementing the trapezoidal membership function.

Fuzzy Rule

fuzzylogic.h

```
// Fuzzy Rule ///////////////////////////////////////////////////////////

typedef   struct {

   short     inp_index[MAX_NO_OF_INPUTS],  //input index
             inp_fuzzy_set[MAX_NO_OF_INPUTS], //input fuzzy set index
             out_fuzzy_set;  //output index

} rule;
```

Define a structure for holding the components of a fuzzy rule.

Note that we are only storing the indices of the components of a rule.

inp_index[0]: **in_angle**

inp_index[1]: **in_distance**

out_fuzzy_set

If (**angle** is **small**) AND (**distance** is **VERY_FAR**) **Then** output is **VERY_FAST**.

inp_fuzzy_set[0]: **in_small**

inp_fuzzy_set[1]: **in_very_far**

Fuzzy System

fuzzylogic.h

Define a structure for holding the complete description of a **fuzzy inference system**.

Here we have a structure implementing a Zero-Order Sugeno Fuzzy inference system.

```
// The complete Fuzzy System /////

typedef  struct fuzzy_system_rec {

    bool allocated;

    trapezoid   inp_mem_fns [MAX_NO_OF_INPUTS] [MAX_NO_OF_INP_REGIONS];

    rule*  rules; //note that we need to allocate memory for the rules

    int     no_of_inputs,          //number of inputs
            no_of_inp_regions,  //number of fuzzy sets associated with each input
            no_of_rules,           //number of rules
            no_of_outputs;        //number of fuzzy outputs

    float  output_values[MAX_NO_OF_OUTPUT_VALUES]; //the values of the outputs

};
```

Defining the Fuzzy Rule

If (**angle** is small) **AND** (**distance** is VERY_FAR) **Then** output is **VERY_FAST**.

```
fuzzy_system_rec *fl;
//…allocate memory
//…initialise fuzzy parameters

    fl->rules[0].inp_index[0] = in_angle;
    fl->rules[0].inp_index[1] = in_distanc

    fl->rules[0].inp_fuzzy_set[0] = in_sm
    fl->rules[0].inp_fuzzy_set[1] = in_ver
    fl->rules[0].out_fuzzy_set = out_very
```

How to customise a fuzzy rule?

Here is an example of how we can define a rule with two inputs.

```
// Fuzzy Rule ////////////////////////////////////////
typedef   struct {
  short    inp_index[MAX_NO_OF_INPUTS],
           inp_fuzzy_set[MAX_NO_OF_INPUTS],
           out_fuzzy_set;  //output index
} rule;
```

SMALL	Med Speed	Fast Sp		y Fast
MEDIUM	Slow Speed	Med Speed	Fast Speed	
LARGE	Very Slow	Slow Speed	Sl	

Defining the Fuzzy Rule

Main.cpp

inp_index[0]: **in_angle** inp_index[1]: **in_distance** **out_fuzzy_set**

If (**angle** is small) **AND** (**distance** is VERY_FAR) **Then** output is **VERY_FAST**.

```
fuzzy_system_rec *fl;
//…allocate memory
//…initialise fuzzy parameters

fl->rules[0].inp_index[0] = in_angle;
fl->rules[0].inp_index[1] = in_distance;

fl->rules[0].inp_fuzzy_set[0] = in_small;
fl->rules[0].inp_fuzzy_set[1] = in_very_far;
fl->rules[0].out_fuzzy_set = out_very_fast;
```

We are only storing the indices in the rule.

```
// Fuzzy Rule ////////////////////////////////////////////
typedef   struct {
  short    inp_index[MAX_NO_OF_INPUTS],
           inp_fuzzy_set[MAX_NO_OF_INPUTS],
           out_fuzzy_set;  //output index
} rule;
```

FAMM

	NEAR	*FAR*	*VERY FAR*
SMALL	Med Speed	Fast Speed	Very Fast
MEDIUM	Slow Speed	Med Speed	Fast Speed
LARGE	Very Slow	Slow Speed	Slow Speed

Membership Function

If (**angle** is small) AND (**distance** is VERY_FAR) Then output is **VERY_FAST**.

fl->inp_mem_fns[variable_index][fuzzy_set]

fl->rules[i].inp_index[0] = in_angle;
fl->rules[i].inp_index[1] = in_distance

fl->rules[0].inp_fuzzy_set[0] = **in_sma**
fl->rules[0].inp_fuzzy_set[1] = in_very
fl->rules[0].out_fuzzy_set = **out_very**

How to specify which **membership function** we want to use?

Using the index of the rule input, and index of the fuzzy set, we can easily identify the membership function.

SMALL	Med Speed	Fast Speed	Fast
MEDIUM	Slow Speed	Med Speed	Fast Speed
LARGE	Very Slow	Slow Speed	Sl

Membership Function

inp_index[0]: **in_angle**

inp_index[1]: **in_distance**

out_fuzzy_set

If (**angle** is small) AND (**distance** is VERY_FAR) Then output is **VERY_FAST**.

fl->**inp_mem_fns****[variable_index][fuzzy_set]**

fl->rules[i].inp_index[0] = in_angle;
fl->rules[i].inp_index[1] = in_distance;

fl->rules[0].inp_fuzzy_set[0] = **in_small**;
fl->rules[0].inp_fuzzy_set[1] = in_very_far;
fl->rules[0].out_fuzzy_set = **out_very_fast**;

	NEAR	*FAR*	*VERY FAR*
SMALL	Med Speed	Fast Speed	Very Fast
MEDIUM	Slow Speed	Med Speed	Fast Speed
LARGE	Very Slow	Slow Speed	Slow Speed

Membership Function data structure

Declaration:

trapezoid **inp_mem_fns** [MAX_NO_OF_INPUTS] [MAX_NO_OF_INP_REGIONS];

inp_mem_fns is a structure containing all the parameters of a membership function.

fl->inp_mem_fns[variable_index][fuzzy_set]

```
fl->rules[i].inp_index[0] = in_angle;
fl->rules[i].inp_index[1] = in_distance;
```

```
fl->rules[0].inp_fuzzy_set[0] = in_small;
fl->rules[0].inp_fuzzy_set[1] = in_very_far;
fl->rules[0].out_fuzzy_set = out_very_fast;
```

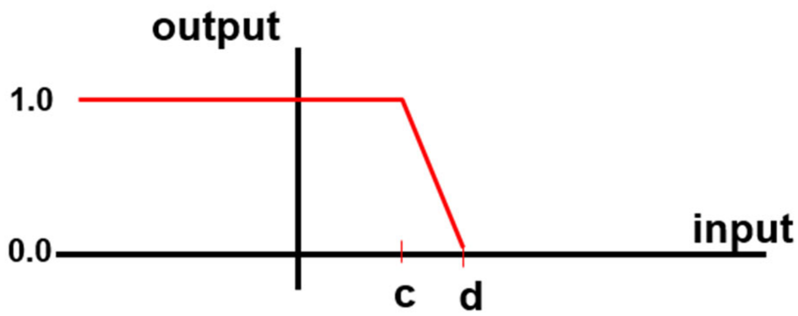
Membership Function Initialisation

inp_mem_fns is a structure containing all the parameters of a membership function.

The **init_trapz()** function is used to initialize **inp_mem_fns**.

```
fl->inp_mem_fns[in_angle][in_small] =
        init_trapz(0.0f, 0.0f, 14.0f, 20.0f, left_trapezoid);
```

```
fl->rules[i].inp_index[0] = in_angle;
fl->rules[i].inp_index[1] = in_distance;

fl->rules[0].inp_fuzzy_set[0] = in_small;
fl->rules[0].inp_fuzzy_set[1] = in_very_far;
fl->rules[0].out_fuzzy_set = out_very_fast;
```



Calculation of degree of Membership

inp_mem_fns is a structure containing all the parameters of a membership function.

The **trapz** function calculates the **actual degree of membership** of a given input value in a fuzzy set.

which input

which membership function

trapz(inputs[variable_index],fl.inp_mem_fns[variable_index][fuzzy_set])

fl->rules[i].inp_index[0] = in_angle;
fl->rules[i].inp_index[1] = in_distance;

fl->rules[0].inp_fuzzy_set[0] = in_small;
fl->rules[0].inp_fuzzy_set[1] = in_very_far;
fl->rules[0].out_fuzzy_set = **out_very_fast**;

Calculation of degree of Membership

fuzzylogic.cpp

which input

which membership function

trapz(inputs[variable_index],fl.inp_mem_fns[variable_index][fuzzy_set])

```
float trapz (float x, trapezoid trz) {
  switch (trz.tp) {

    case left_trapezoid: //opening to the left
            if (x <= trz.c)
              return 1.0;
            if (x >= trz.d)
              return 0.0;
            /* c < x < d */
            return trz.r_slope * (x - trz.d);


    case right_trapezoid: //opening to the right
            if (x <= trz.a)
              return 0.0;
            if (x >= trz.b)
              return 1.0;
            /* a < x < b */
            return trz.l_slope * (x - trz.a);
```

```
    case regular_trapezoid:
            if ((x <= trz.a) || (x >= trz.d))
              return 0.0;
            if ((x >= trz.b) && (x <= trz.c))
              return 1.0;
            if ((x >= trz.a) && (x <= trz.b))
              return trz.l_slope * (x - trz.a);
            if ((x >= trz.c) && (x <= trz.d))
              return  trz.r_slope * (x - trz.d);

            }  /* End switch  */

  return 0.0;  /* should not get to this point */
}  /* End function */
```

Fuzzy System – from fuzzification to defuzzification

File: **fuzzylogic.cpp**

```
float fuzzy_system ( float inputs[ ], fuzzy_system_rec fz) {
   int i, j;
   short variable_index, fuzzy_set;
   float sum1 = 0.0f, sum2 = 0.0f, weight;
   float m_values[MAX_NO_OF_INPUTS];

   for (i = 0; i < fz.no_of_rules; i++) {
      for (j = 0; j < fz.no_of_inputs; j++) {
                    variable_index = fz.rules[i].inp_index[j];
                    fuzzy_set = fz.rules[i].inp_fuzzy_set[j];
                    m_values[j] = trapz(inputs[variable_index], fz.inp_mem_fns[variable_index][fuzzy_set]);
       } /* end j */

      weight = min_of (m_values, fz.no_of_inputs);
      sum1 += weight * fz.output_values[ fz.rules[i].out_fuzzy_set ];
      sum2 += weight;

   } /* end i */

   if (fabs(sum2) < TOO_SMALL) { // TOO_SMALL = 1e-6
      cout << "\r\nFLPRCS Error: sum2 in fuzzy_system is 0." << endl;
      //exit(1);
      return 0.0;
   }

   return (sum1/sum2);
}
```

which input

which fuzzy set

How to use the Fuzzy Logic Engine?

Fuzzy System Development

1. Declare a global variable for the fuzzy system.

File: **Main.cpp**

e.g. **fuzzy_system_rec** g_fuzzy_system;

Define the maximum number of Fuzzy Sets to be used.

File: **fuzzylogic.h**

```
#define  MAX_NO_OF_INP_REGIONS  7
```

e.g. **7** Maximum Number of Fuzzy Sets allowed

Fuzzy System Development

2. Initialise the fuzzy system.

File: **fuzzylogic.cpp**

```
void   initFuzzySystem (fuzzy_system_rec*   fl) {

   fl->no_of_inputs = 2;  //inputs are handled 2 at a time only
   fl->no_of_inp_regions = 7; //number of fuzzy sets per input
   fl->no_of_rules = 49*2;          //number of rules
//----
   fl->output_values [out_small_push_left] = -15f;   //-15 Newtons
   fl->output_values [out_large_push_left] = -100f; //-100 Newtons

   //… and so on…
   …
   fl->rules = new rule [fl->no_of_rules];  //allocate memory for the rules

   initFuzzyRules (fl); //initialise the rules; this is user-defined
   initMembershipFunctions(fl); //initialise the membership functions; this is user-
                                //defined
}
```

Fuzzy System Development

3. Initialise fuzzy rules.

File: **fuzzylogic.cpp**

```
void initFuzzyRules (fuzzy_system_rec*  fl) {
  int i;
  for (i = 0;i < fl->no_of_rules;i++) {
     //(*fl).rules[i].inp_index[0] = in_angle; //alternatively
     fl->rules[i].inp_index[0] = in_angle;
     fl->rules[i].inp_index[1] = in_angle_dot;
  }

  fl->rules[0].inp_fuzzy_set[0] = in_negatively_small;
  fl->rules[0].inp_fuzzy_set[1] = in_falling_to_left_fast;
  fl->rules[0].out_fuzzy_set = out_large_push_left;

// define the other remaining rules next
// and so on...

}
```

Rule #0

This is an example only. The inputs and rule parameters will depend on your system design.

Fuzzy System Development

3. Initialise fuzzy rules.

File: **fuzzylogic.cpp**

```
void initFuzzyRules (fuzzy_system_rec*  fl) {
  int i;
  for (i = 0;i < fl->no_of_rules;i++) {
     //(*fl).rules[i].inp_index[0] = in_X; //alternatively
     fl->rules[i].inp_index[0] = in_X;
     fl->rules[i].inp_index[1] = in_Y;
  }

  fl->rules[0].inp_fuzzy_set[0] = in_NM;
  fl->rules[0].inp_fuzzy_set[1] = in_PM;
  fl->rules[0].out_fuzzy_set = out_NS;

// define the other remaining rules next
// and so on...

}
```

Rule #0

$X = A\theta_t + B\dot{\theta}_t$

$Y = Cx + D\dot{x}$	NM	NS	ZR	PS	PM
PM	NS		PS		(PL)
PS		NS		PM	
ZR	NM		ZR		PM
NS		NM		PS	
NM	(NL)		NS		PS

Yamakawa's strategy

Fuzzy System Development

4. Initialise membership functions.

File: **fuzzylogic.cpp**

```
void   initMembershipFunctions ( fuzzy_system_rec*   fl ) {

  //angle
  fl->inp_mem_fns[in_angle][in_negatively_large] = init_trapz(14.0f,20.0f,0.0f,0.0f,
                                                              left_trapezoid);
  fl->inp_mem_fns[in_angle][in_negatively_medium] = init_trapz(14.0f,20.0f,34.0f,40.0f,
                                                              regular_trapezoid);
 //...
  fl->inp_mem_fns[in_angle][in_positively_large] = init_trapz (34.0f, 40.0f, 0.0f, 0.0f,
                                                              right_trapezoid);

  //angular velocity
  //...
  //...
  //...
  //and so on...

}
```

which fuzzy set

Inverted Pendulum

4. Modify runInvertedPendulum ().

File: Main.**cpp**

```
void   runInvertedPendulum () {
  //...
  initFuzzySystem(&g_fuzzy_system);
  while((GetAsyncKeyState(VK_ESCAPE)) == 0 ) { //while ESC key is not pressed
    //...
     inputValues [ in_angle ] = prevState.angle;
     inputValues[ in_angle_dot ] = prevState.angle_dot;
     prevState.Force= fuzzy_system(inputValues, g_fuzzy_system);
     //...  Calculate new state of the world
     newState.angle_double_dot = calc_angular_acceleration(prevState);
     //and so on...
     cart.setX(newState.x);
     rod.setX(newState.x);
     rod.setAngle(newState.angle);
     cart.draw();   rod.draw();
}
   free_fuzzy_rules(&g_fuzzy_system);
  //and so on...
}
```

Specify the inputs and fuzzy system structure, then pass them to the fuzzy_system() function.

Skeleton of Simulation

void runInvertedPendulum()

```
initPendulumWorld();
initFuzzySystem(&g_fuzzy_system);

while((GetAsyncKeyState(VK_ESCAPE)) == 0 ) {

        setactivepage(page);
        cleardevice();

        senseEnvironment (inputs);

        F = fuzzy_system(inputs, g_fuzzy_system);

        updateWorld(newWorldState)
        drawCartAndRod(newWorldState)

        setvisualpage(page);
        toggle page  //switch to another page
}
```

Input and Output

INPUTS

1. Angle of the rod with respect to the vertical axis (in radians)
2. Angular velocity of the rod (in radians per second)
3. Cart position (in meters)
4. Cart velocity (in meters per second)

OUTPUT

OUTPUT

1. Force to apply (in Newtons)

World State

```
struct WorldStateType{
	void init(){
		x=0.0;
		x_dot=0.0;
		x_double_dot = 0.0;
		angle = 0.0;
		angle_dot = 0.0;
		angle_double_dot = 0.0;
		F = 0.0;
	}

	float x;
	float x_dot;
	float x_double_dot;
	float angle;
	float angle_dot;
	float angle_double_dot;

float const mb=0.1; // mass of broom
float const g=9.8; // pull of gravity
float const m=1.1; // mass of cart & broom
float const l=0.5; // length of broom from
//pivot point to centre of mass

float F;

};
```

```
WorldStateType prevState, newState;
```

Dynamics of the System

```
float calc_angular_acceleration( const WorldStateType& s){
    float a_double_dot=0.0;

    a_double_dot = (s.m * s.g * sin(s.angle) - (cos(s.angle)
                        * (s.F + ((s.mb) * s.l * s.angle_dot * s.angle_dot * sin(s.angle) ) ) )  )
                       / (  ((4/3)*s.m * s.l) - (s.mb * s.l * cos(s.angle) * cos(s.angle)));

    return a_double_dot;
}
```

```
float calc_horizontal_acceleration( const WorldStateType& s){
     float x_double_dot=0.0;

     x_double_dot = ( s.F + s.mb * s.l * (s.angle_dot * s.angle_dot)* sin(s.angle) -
                         s.angle_double_dot * cos(s.angle)   )/ s.m;

    return x_double_dot;
}
```

void runInvertedPendulum()

```
//*****************************************************************
//Set the initial angle of the pole with respect to the vertical
prevState.angle = 8 * (3.14/180);  //initial angle  = 8 degrees

initFuzzySystem(&g_fuzzy_system);

while((GetAsyncKeyState(VK_ESCAPE)) == 0 ) {

 setactivepage(page);
 cleardevice();
 drawInvertedPendulumWorld();

 //retrieve inputs
 inputs[in_theta] = prevState.angle;
 inputs[in_theta_dot] = prevState.angle_dot;
 inputs[in_x] = prevState.x;
 inputs[in_x_dot] = prevState.x_dot;

 //1) Enable this only after your fuzzy system has been completed already.
 //Remember, you need to define the rules, membership function parameters and rule outputs.
 //prevState.F = fuzzy_system(inputs, g_fuzzy_system); //call the fuzzy controller

 externalForce=0.0;
 externalForce = getKey(); //manual operation

 if(externalForce != 0.0)
    prevState.F = externalForce;
//-----------------------------------------------------------------------------
//continued…
```

Force to apply.

void runInvertedPendulum()

```
//--------------------------------------------------------------------------
 // ************************************************************************
 // BEGIN - DYNAMICS OF THE SYSTEM
 //Calculate the new state of the world
 newState.angle_double_dot = calc_angular_acceleration(prevState);
 newState.angle_dot = prevState.angle_dot + (h * newState.angle_double_dot);
 newState.angle = prevState.angle + (h * newState.angle_dot);
 newState.F = prevState.F;
 newState.x_double_dot = calc_horizontal_acceleration(prevState);
 newState.x_dot = prevState.x_dot + (h * newState.x_double_dot);
 newState.x = prevState.x + (h * newState.x_dot);
 prevState.x = newState.x;
 prevState.angle = newState.angle;
 prevState.x_dot = newState.x_dot;
 prevState.angle_dot = newState.angle_dot;
 prevState.angle_double_dot = newState.angle_double_dot;
 prevState.x_double_dot = newState.x_double_dot;
 //--------------------------
 cart.setX(newState.x);
 rod.setX(newState.x);
 rod.setAngle(newState.angle);
 cart.draw();
 rod.draw();
 // END - DYNAMICS OF THE SYSTEM
 // ************************************************************************
 //--------------------------------------------------------------------------
 displayInfo(newState);
 setvisualpage(page);
 page = !page;  //switch to another page
}
/2) Enable this only after your fuzzy system has been completed already.
//free_fuzzy_rules(&g_fuzzy_system);
```

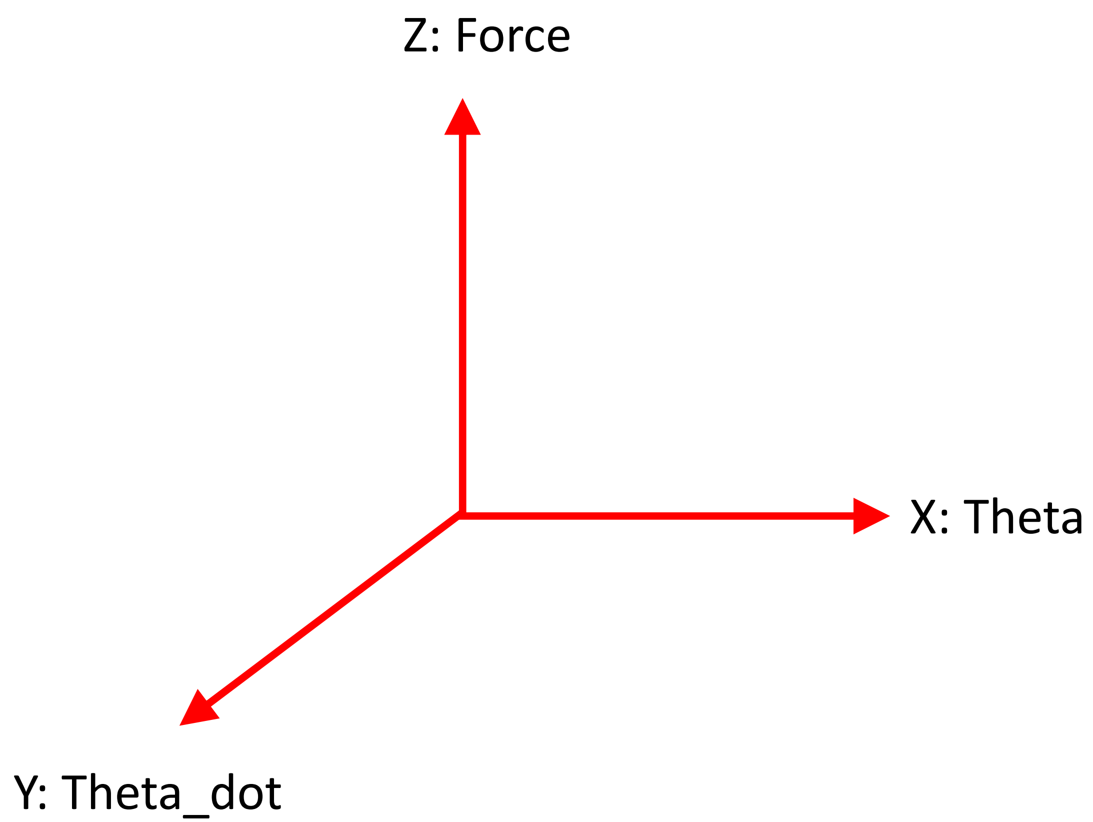
Physics equations and Euler’s method (please do not modify these statements)

Control surface generation

int main()

```
int main() {

    int graphDriver = 0,graphMode = 0;

    initgraph(&graphDriver, &graphMode, "", 1280, 1024); // Start Window
    clearDataSet();
    try{
          runInvertedPendulum();

          //3) Enable this only after your fuzzy system has been completed already.
          //generateControlSurface_Angle_vs_Angle_Dot();

          //4) Enable this only after your fuzzy system has been completed already.
          //saveDataToFile("data_angle_vs_angle_dot.txt");

    }
    catch(...){
      cout << "Exception caught!\n";
    }
return 0;
}
```

Control surface generation



X: Theta [-40 deg., 40 deg] = [-0.7 rad., 0.7 rad.]

Y: Theta_dot = [-3 rad./sec, 3 rad./sec]

World Coordinate to Device Coordinate Transformation

for defining the system of coordinates, scaling and zooming-in or out.