

Laporan dokumentasi penggerjaan tugas Final Project Magang Bayucaraka

Farrel Ganendra | 5024231036

Tugas diawali dengan membuat folder sebagai workspace bernama FPMagangBayu24, kemudian navigasi ke folder tersebut di terminal dan masukkan command “code .” untuk membuat vscode dengan folder yang kita pilih sebagai workspace. Kemudian, pada terminal vscode yang sudah log in ke akun github, masukkan command “git init” untuk menginisialisasi repositori baru, masuk ke website github dan buat repository privat baru, copy link https repository dan kembali ke terminal vscode dan lakukan

```
git add remote origin https://github.com/Kak-Palel/FPMagangBayu24.git
```

Kemudian, buat folder src dengan command “mkdir src” dan lalu navigasi ke folder tersebut. Kemudian buat dua package vision dan control dengan command berikut.

```
ros2 pkg create --build-type ament_python vision
```

```
ros2 pkg create --build-type ament_cmake control
```

Kemudian pada package vision, didalam folder vision, buat file baru bernama visioner.py dan copy contoh yang diberikan pada wiki ros2

```
import rclpy
from rclpy.node import Node

from std_msgs.msg import String

class MinimalPublisher(Node):

    def __init__(self):
        super().__init__('minimal_publisher')
        self.publisher_ = self.create_publisher(String, 'topic', 10)
        timer_period = 0.5 # seconds
        self.timer = self.create_timer(timer_period, self.timer_callback)
        self.i = 0

    def timer_callback(self):
        msg = String()
        msg.data = 'Hello World: %d' % self.i
        self.publisher_.publish(msg)
        self.get_logger().info('Publishing: "%s"' % msg.data)
        self.i += 1

def main(args=None):
    rclpy.init(args=args)

    minimal_publisher = MinimalPublisher()

    rclpy.spin(minimal_publisher)

    # Destroy the node explicitly
    # (optional - otherwise it will be done automatically
    # when the garbage collector destroys the node object)
    minimal_publisher.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Code tersebut hanya membuat sebagai publisher, langkah pertama yang ingin saya lakukan adalah untuk mendirikan komunikasi antara 2 node, sehingga kita perlu memodifikasi file ini sehingga dapat menjadi node yang dapat mempublish dan juga menerima suatu informasi. Pertama dengan mengubah nama class nya menjadi 'VisionerTicTacToe'. Kemudian pada constructor class, mengubah topic nya menjadi 'ttt_state' yang akan menjadi topik tentang keadaan grid dari tic tac toe. Selain itu, juga menginisialisasi listener yang mendengarkan topik 'ttt_moves' yang akan menjadi topik tentang gerakan bot.

```
#membuat subscription untuk menerima perintah gerakan bot
self.subscription = self.create_subscription(
    String,
    'ttt_moves',
    self.listener_callback,
    10)
self.subscription #ini biar ga di warning bjir
```

Tentunya kita juga perlu fungsi callback yang menerima pesan gerakan dari bot.

```
def listener_callback(self, msg):
    self.get_logger().info('I heard: "%s"' % msg.data)
```

Jangan lupa add dependencies terhadap rclpy dan std_msgs pada package.xml dan tambahkan entry point pada setup.py dengan nama node bot_vision.

Setelah itu kita dapat beralih dulu package control, pada folder src, buat file baru bernama processor.cpp dan copy contoh publisher pada wiki ros2.

```
#include <chrono>
#include <functional>
#include <memory>
#include <string>

#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/string.hpp"

using namespace std::chrono_literals;

/* This example creates a subclass of Node and uses std::bind() to register a
 * member function as a callback from the timer. */

class MinimalPublisher : public rclcpp::Node
{
public:
    MinimalPublisher()
    : Node("minimal_publisher"), count_(0)
    {
        publisher_ = this->create_publisher<std_msgs::msg::String>("topic", 10);
        timer_ = this->create_wall_timer(
            500ms, std::bind(&MinimalPublisher::timer_callback, this));
    }

private:
    void timer_callback()
    {
        auto message = std_msgs::msg::String();
        message.data = "Hello, world! " + std::to_string(count_++);
        RCLCPP_INFO(this->get_logger(), "Publishing: '%s'", message.data.c_str());
        publisher_->publish(message);
    }
    rclcpp::TimerBase::SharedPtr timer_;
    rclcpp::Publisher<std_msgs::msg::String>::SharedPtr publisher_;
    size_t count_;
};

int main(int argc, char * argv[])
{
    rclcpp::init(argc, argv);
    rclcpp::spin(std::make_shared<MinimalPublisher>());
    rclcpp::shutdown();
    return 0;
}
```

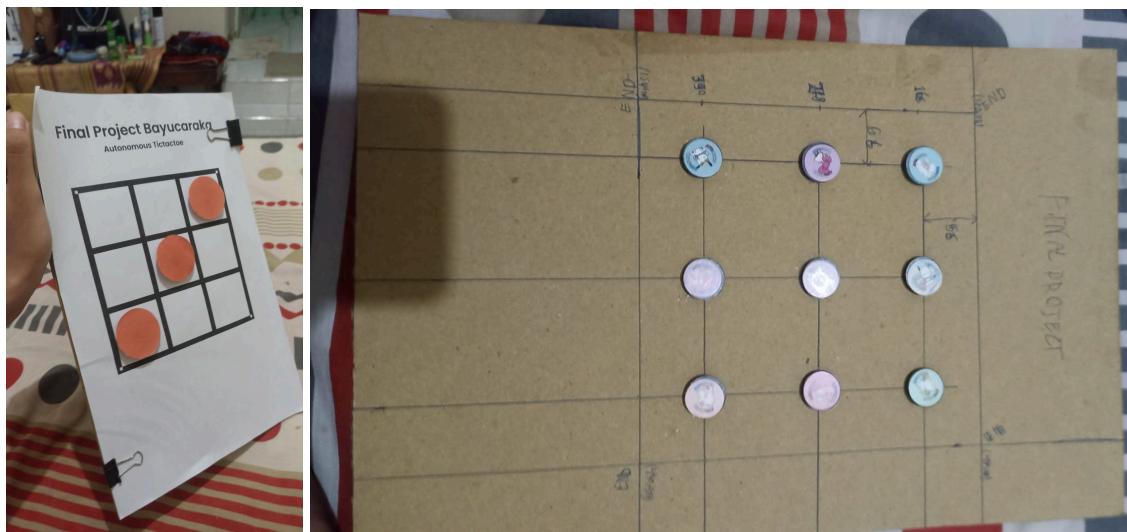
Sama seperti sebelumnya, kita perlu memodifikasi code ini sehingga dapat menerima sekaligus mengirim pesan di topik yang berbeda. Untuk itu, pertama kita ubah nama classnya menjadi ControllerTicTacToe, lalu mengubah topik publisher ke 'ttt_moves'. Selain itu, kita juga perlu menginisiasi listener yang mendengar topik 'ttt_state' pada constructor class.

```
//inisiasi subscription untuk menerima kondisi grid dari node vision
subscription_ = this->create_subscription<std_msgs::msg::String>(
    "ttt_state", 10, std::bind(&ControllerTicTacToe::topic_callback, this, _1));
```

Lalu buat method sebagai fungsi callback.

```
void topic_callback(const std_msgs::msg::String::SharedPtr msg) const
{
    RCLCPP_INFO(this->get_logger(), "I heard: '%s'", msg->data.c_str());
}
rclcpp::Subscription<std_msgs::msg::String>::SharedPtr subscription_;
```

Tambahkan dependencies terhadap rclcpp dan std_msgs pada CMakeLists.txt dan package.xml, dan install node dengan nama node bot_control. Build kedua package dan jalankan kedua node untuk memastikan komunikasi antar node berjalan dengan baik. Setelah itu. Untuk mempermudah proses penggeraan, saya membuat papan tictactoe dengan bidak merah yang menempel pada papan tersebut dengan magnet.



Masalah selanjutnya adalah bagaimana node bot_vision dapat membaca posisi grid lewat camera, dan mengubahnya ke bentuk string untuk dapat dikirim ke node bot_control. Untuk menyelesaikan ini, pertama kita import cv2 sebagai cv, kemudian pada bagian constructor, kita buka kamera set source opencv ke kamera kita dengan command ‘self.cap = cv.VideoCapture(0)’. Kemudian, karena method time ‘timer_callback’ dipanggil 30 kali per detik, kita dapat membuat method lain yang akan dipanggil didalam method publisher ini.

```
#method publisher yang hanya mempublish apabila user telah memberi aba aba untuk berpindah gantian
def timer_callback(self):
    msg = String()
    msg.data = self.readState()
    if msg.data != 'zonk':
        self.publisher_.publish(msg)
        print(msg.data)
        print('move sent, processing...')
```

Perhatikan bahwa method ini walaupun terpanggil 30 kali per detik, dia hanya akan mempublish apabila di salah satu iterasinya, data yang di return oleh method ‘readState()’ bukan ‘zonk’. Hal ini dikarenakan kita tidak perlu mempublish kondisi grid 30 kali per detik, kita hanya perlu mempublish keadaan grid ketika kita membuat perubahan pada grid.

Untuk membaca grid, kita pertama perlu membaca data dari capture kamera, kemudian kita copy sehingga menjadi 2 gambar. Gambar pertama untuk ditampilkan pada user dan juga yang akan digambari bidak biru milik bot, kemudian gambar yang kedua akan digunakan untuk mendeteksi bidak merah milik user. Langkah yang pertama dapat kita ambil adalah dengan melakukan mask warna merah pada gambar original, hal ini dilakukan dengan mengonversi warna gambar dari BGR ke HSV dengan method kelas cv ‘cv.cvtColor()’ dan menggunakan fungsi callback ‘cv.COLOR_BGR2HSV’ di parameter keduanya. Terdapat suatu masalah, Pada roda HSV, warna merah terdapat di dua area, yaitu pada 15 derajat pertama dan juga 15 derajat terakhir. Hal ini dapat kita selesaikan dengan melakukan mask pada 2 gambar copy yang terpisah dan lalu melakukan penambahan dari hasil nya.

```
#mask untuk nilai hsv merah yang pertama
upper_range = np.array([15, 255, 255])
lower_range = np.array([0, 100, 100])
mask1 = cv.inRange(mask1, lower_range, upper_range)

#mask untuk nilai hsv merah yang kedua
upper_range = np.array([179, 255, 255])
lower_range = np.array([155, 100, 100])
mask2 = cv.inRange(mask2, lower_range, upper_range)

#gabungkan kedua mask untuk mendapatkan semua warna merah
processFrame = mask1 + mask2
```

Dengan menggunakan method ‘cv.imshow()’, kita dapat melihat hasil dari proses masking yang kita lakukan, ternyata lingkaran bidak yang telah kita buat sebelumnya tidak terisolasi dengan sempurna, bentuknya bukan lingkaran sempurna. Hal ini diakibatkan oleh rendahnya resolusi kamera (640x480) dan juga buruknya pencahayaan. Untuk meminimalisir masalah ini, kita dapat apply blur sebelum melakukan mask. Saya mencoba semua jenis smoothing image mulai dari ‘cv.blur()’, ‘cv.medianBlur()’, ‘cv.gaussianBlur()’, dan juga ‘cv.bilateralFilter()’, hasilnya ‘cv.medianBlur()’ dan ‘cv.bilateralFilter()’ menghasilkan hasil yang terbaik. ‘cv.bilateralFilter()’ menghasilkan lingkaran yang lebih tajam namun tidak terisi putih dengan sempurna, oleh karena itu, saya memutuskan untuk menggunakan ‘cv.medianBlur()’.

Kemudian Setelah itu, kita perlu untuk menemukan semua lingkaran bidak user pada gambar yang telah di proses, hal ini dapat kita lakukan dengan fungsi bawaan opencv bernama ‘cv.HoughCircles()’. Namun, lingkaran yang terdeteksi sering tidak akurat dan saya kebingungan untuk menentukan nilai yang benar untuk parameter param2 (accumulator threshold untuk pusat pusat lingkaran pada tahap deteksi). Untuk mengatasi hal ini, Saya membuat window untuk menampilkan hasil deteksi dan lalu membuat trackbar yang mengubah variabel untuk param2 ini sehingga saya dapat menentukan value yang terbaik secara realtime. Pada kondisi saya, 15 merupakan value yang terbaik sehingga saya set default value nya menjadi 15.

Langkah selanjutnya adalah untuk menentukan lingkaran yang telah terdeteksi merupakan bidak yang terdapat di kolom ke berapa pada grid tictactoe. Sebelum itu, program perlu mengetahui batasan batasan tiap kolom pada grid dan juga index dari gridnya, hal ini dapat dilakukan membuat array yang menyimpan batas koordinat kiri, kanan, atas, bawah, dan juga index kolom.

```
#batas batas tiap kolom di grid dan juga index kolomnya
gridPos = np.array([
    [150, 263, 110, 223, 0],
    [264, 376, 110, 223, 1],
    [377, 490, 110, 223, 2],
    [150, 263, 224, 336, 3],
    [264, 376, 224, 336, 4],
    [377, 490, 224, 336, 5],
    [150, 263, 337, 450, 6],
    [264, 376, 337, 450, 7],
    [377, 490, 337, 450, 8]])
```

Lalu kita dapat membuat array bernama ‘gridState’ yang beranggotakan 9 integer, array ini merepresentasikan kondisi dari grid dengan ‘0’ berarti kosong, ‘1’ berarti bidak user, ‘2’ berarti bidak bot. Kemudian kita dapat mengiterasi semua lingkaran yang terdeteksi dan mengecek posisi mereka berada pada di grid ke berapa dan lalu memasukkannya ke array ‘gridState’.

```
result = getstate(userFrame(obj), method)(*args, **kwargs)
TypeError: loop of ufunc does not support argument 0 of type NoneType which
has no callable rint method
```

Ketika kita panggil method pembaca gridstate ini, kita mendapat error seperti berikut, hal ini terjadi karena kadang program tidak mendeteksi lingkaran sama sekali sehingga array circles kosong. Mengiterasi array yang kosong akan melemparkan error. Hal ini dapat kita selesaikan dengan mudah menggunakan if statement.

```
#gambar posisi bidak player yang terdeteksi dan assign sebagai gerakan yang telah dibuat oleh player
if circles is not None:
    circlesInt = np.round(circles[0, :]).astype("int")
    for (x, y, r) in circlesInt:
        cv.circle(drawnFrame, (x, y), r, (255, 0, 0), 2)
        for i in gridPos:
            if i[0] <= x and x <= i[1] and i[2] <= y and y <= i[3] and gridState[i[4]] != 2:
                gridState[i[4]] = 1
```

Langkah terakhir adalah dengan menunjukkan hasil yang kita peroleh ke user dan mengonversi pesannya menjadi string supaya dapat dikirim ke topik ‘ttt_state’ oleh publisher.

```
#tunjukkan gambar utama dan juga gambar yang dilihat oleh komputer untuk keperluan kalibrasi
cv.imshow("olel ngeri", processFrame)
cv.imshow("ni olel", drawnFrame)

#konversi kondisi grid dari bentuk array ke string
msg = ''
for i in gridState:
    msg += str(i)

if cv.waitKey(1) == ord('x'):          #giliran selanjutnya
    return msg

return 'zonk'
```

User dapat menekan tombol x untuk memberitahu program bahwa user telah melakukan gerakannya dan kondisi dari grid siap dikirim ke node ‘bot_control’.

Kondisi grid telah dikirim dan diterima oleh node ‘bot_control’. Langkah selanjutnya adalah bagaimana bot dapat menentukan gerakan yang ingin dimainkan dengan benar. Strategi awal yang saya pikirkan adalah dengan memilih random kecuali jika player akan menang, bot akan berusaha menghadang player untuk menang. Sayangnya, user dapat menggunakan strategi tertentu sehingga memiliki dua jalan untuk menang dan bot hanya dapat menghadang salah satunya.

Kemudian saya berfikir untuk mengiterasi semua kemungkinan dan memberikan skor 1 setiap bot menang, -1 ketika user menang, dan 0 ketika seri. Skor ini di kumpulkan dan lalu bot memilih yang paling tinggi, sayangnya gerakan yang menyebabkan bot langsung menang hanya akan memiliki skor 1 sedangkan gerakan yang bertele-tele akan memiliki skor lebih tinggi.

Saya pun menemukan strategi minimax dan strategi alphabeta di internet (<https://cwoebker.com/posts/tic-tac-toe>). Strategi ini mengiterasi semua kemungkinan secara rekursif, mengambil asumsi bahwa user akan selalu bermain

optimal, dan memaksimalkan probabilitas kemenangan dari bot. Saya pun memutuskan untuk menggunakan algoritma alphabeta. Untuk mengimplementasikan algoritma ini, pertama kita perlu beberapa hal yaitu:

- Method ‘string2vec()’ untuk mengubah pesan gridstate yang diterima dari bentuk string ke vector supaya lebih mudah untuk diproses

```
//method untuk mengubah string posisi grid ke vector
std::vector<int> string2vec(std::string strBoard) const
{
    std::vector<int> vecBoard;
    for(int i = 0; i < 9; i++)
    {
        vecBoard.push_back(std::stoi(strBoard.substr(i, 1), nullptr, 10));
    }
    return vecBoard;
}
```

- Variabel pada scope class bersifat privat yang akan menyimpan gerakan final yang mau dikirim, -1 sebagai nilai default, jika nilainya berubah dari -1 ke index grid, maka siap untuk dikirim dan publisher mengubahnya kembali ke -1.
- Method ‘availableMoves()’ untuk menentukan kolom mana saja yang dapat dimainkan

```
//method untuk menentukan kolom grid mana saja yang dapat menjadi gerakan bot
std::vector<int> availableMoves(std::vector<int> boardState)
{
    std::vector<int> moves;
    for(int i = 0; i < 9; i++)
    {
        if(boardState[i] == 0) {moves.push_back(i);}
    }
    return moves;
}
```

- Kombinasi grid yang memenangkan suatu player apabila dipopulasi ketiganya

```
//kombinasi kombinasi kolom grid yang menandakan kemenangan bila di populasi bersamaan
std::vector<std::vector<int>> winCombination =
{
    {0, 1, 2},
    {3, 4, 5},
    {6, 7, 8},
    {0, 3, 6},
    {1, 4, 7},
    {2, 5, 8},
    {0, 4, 8},
    {2, 4, 6}
};
```

- Method ‘checkWin()’ untuk mengecek apakah suatu player telah menang atau belum

```

//method untuk mengecek suatu player telah menang atau belum
bool checkWin(std::vector<int> boardState, int player)
{
    for (uint64_t i = 0; i < winCombination.size(); i++)
    {
        if
        (
            boardState[winCombination[i][0]] == player &&
            boardState[winCombination[i][1]] == player &&
            boardState[winCombination[i][2]] == player
        )
        {return true;}
    }
    return false;
}

```

- Implementasi dari algoritma alphabeta itu sendiri

```

//method implementasi algoritma alpha-beta
int alphaBeta(std::vector<int> boardState, int playing, int alpha, int beta)
{
    if(checkWin(boardState, 2)) {return 1;}
    else if(checkWin(boardState, 1)) {return -1;}
    else
    {
        for(uint64_t i = 0; i < boardState.size(); i++)
        {
            if(boardState[i] == 0) {break;}
            if(i == 7) {return 0;}
        }

        int score = 0;
        std::vector<int> emptyGrids = availableMoves(boardState);
        for(uint64_t i = 0; i < emptyGrids.size(); i++)
        {
            boardState[emptyGrids[i]] = playing;
            score = alphaBeta(boardState, (playing == 1 ? 2 : 1), alpha, beta);
            boardState[emptyGrids[i]] = 0;
            if(playing == 2)
            {
                if(score > alpha) {alpha = score;}
                if(alpha >= beta) {return beta;}
            }
            else
            {
                if(score < beta) {beta = score;}
                if(beta <= alpha) {return alpha;}
            }
        }
        if(playing == 2) {return alpha;} return beta;
    }
}

```

- Method ‘determine()’ yang memanggil method alphabeta untuk semua gerakan yang dapat dimainkan dan mereturn gerakan pertama yang memiliki index score 1, return -1 sebagai default

```

int determine(std::vector<int> boardState)
{
    int maxScore = -2;
    int maxMove = -1;
    std::vector<int> moves = availableMoves(boardState);
    if (moves.size() == 9) {return 4;}
    for(uint64_t i = 0; i < moves.size(); i++)
    {
        boardState[moves[i]] = 2;
        int tempScore = alphaBeta(boardState, 1, -2, 2);
        std::cout<<"finalScore: "<<tempScore<<std::endl;
        if(tempScore > maxScore)
        {
            maxScore = tempScore;
            maxMove = moves[i];
            if(maxScore == 1) {return maxMove;}
        }
        boardState[moves[i]] = 0;
    }
    return maxMove;
}

```

Dengan kita memiliki semua hal tersebut, kita dapat memanggil proses proses ini ketika node menerima giliran.

```

void topic_callback(const std_msgs::msg::String::SharedPtr msg)
{
    //terima posisi grid
    RCLCPP_INFO(this->get_logger(), "A: '%s'", msg->data.c_str());
    std::string strBoard = msg->data.c_str();
    std::cout<<strBoard<<std::endl;

    //ubah dalam bentuk vector
    std::vector<int> vecBoard = this->string2vec(strBoard);

    //print untuk keperluan debugging
    for(int j = 0; j < 9; j++)
    {
        std::cout<<vecBoard[j]<<" ";
        if((j+1) % 3 == 0){std::cout<<std::endl;}
    }
    std::cout<<std::endl;

    //panggil fungsi determine dan simpan ke finalMove
    finalMove = determine(vecBoard);
    std::cout<<"move: "<<finalMove<<std::endl;
}

rclcpp::Subscription<std_msgs::msg::String>::SharedPtr subscription_;

```

Langkah terakhir untuk node ini adalah dengan mengirim bentuk string dari index kolom yang ingin dimainkan oleh bot apabila semua proses telah selesai (nilai finalMove berubah).

```
//method yang mempublish finalMove dalam bentuk string apabila nilainya berubah dari default valuenya (-1)
void timer_callback()
{
    if(finalMove != -1)
    {
        auto message = std_msgs::msg::String();
        message.data = std::to_string(finalMove);
        publisher_->publish(message);
        finalMove = -1;
    }
}
rclcpp::TimerBase::SharedPtr timer_;
rclcpp::Publisher<std_msgs::msg::String>::SharedPtr publisher_;
size_t count_;
```

Setelah dikirim melalui topik ‘ttt_moves’, node vision pun menerima dan menggambar bidak biru milik bot. Tapi sebelum itu, kita juga memerlukan sebuah array yang menyimpan semua gerakan yang telah dilakukan oleh bot, sehingga pada constructor, kita inisiasi array tersebut dengan

```
self.botMoves = np.array([])
```

Kemudian untuk method callback nya hanya menerima pesan dan menyimpannya pada array ini.

```
#method listener yang menerima perintah gerakan yang telah dikirim oleh node control dan menyimpannya
def listener_callback(self, msg):
    self.get_logger().info('bot move to grid "%s" % msg.data')
    self.botMoves = np.append(self.botMoves, int(msg.data))
```

Terakhir kita dapat menggambar bidak bot dan juga assign gerakan bot ke kondisi grid yang akan dikirim kembali, hal ini dilakukan dengan memodifikasi method ‘readState()’

```
#gambar posisi bidak bot dan assign sebagai gerakan yang telah dibuat oleh bot
if self.botMoves is not None:
    for i in self.botMoves:
        i = int(i)
        gridState[i] = 2
        if 0 <= i <= 2:
            cv.circle(drawnFrame, (150 + 56*(2*i+1), 166), 50, (255, 0, 0), -1)
        elif 3 <= i <= 5:
            cv.circle(drawnFrame, (150 + 56*(2*(i-3)+1), 278), 50, (255, 0, 0), -1)
        elif 6 <= i <= 8:
            cv.circle(drawnFrame, (150 + 56*(2*(i-6)+1), 390), 50, (255, 0, 0), -1)
```

Tictactoe pun sudah dapat dimainkan. Jangan lupa lakukan source ke direktori ros dan juga local repository sebelum menjalankan node. Berikut link video demonstrasi nya

 VID_20240223_210723.mp4

<https://drive.google.com/file/d/1vdE8Q0-BIKTWwl8pdCK1rpX9WA2xEG42/view?usp=sharing>