SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

JEDNODUCHÁ AUTOMATICKÁ MISIA

ZADANIE

2024 **Bc. Maroš Kocúr, Bc. Branislav Hetnerovič**

Obsah

1 Úlohy

Cieľom projektu je vytvoriť autonómny systém na riadenie dronu s využitím ROS, ktorý bude schopný autonómne navigovať v prostredí s prekážkami. Projekt zahŕňa načítanie máp, plánovanie trajektórie a kontrolu pozície dronu na základe zadaných bodov a aktuálnej polohy dronu.

1. **Spracovanie máp**
    - Implementácia alebo nástroj - načítanie máp - **povinné**
    - Implementácia inflácie prekážok - **povinné**
    - Implementácia transformácií máp, rotácií atď. - **ak je potrebné**
2. **Algoritmus vyhľadávania ciest (Path finding)**
    - Implementácia alebo nástroj - algoritmus vyhľadávania ciest - **povinné**
        - Flood fill, RRT, A*...
3. **Plánovanie trajektórie**
    - Implementácia alebo nástroj - optimalizácia trajektórie - **ak je potrebné**
        - Minimálna požiadavka - eliminácia nepotrebných bodov (eliminácia sekvencií horizontálnych, vertikálnych, diagonálnych bodov)
4. **ROS Uzol pre riadenie dronu**
    - Implementácia - načítanie trajektórie - **povinné**
    - Implementácia - úlohy/misie - **povinné**
    - Implementácia - regulátor polohy - **povinné**
5. **Špecifikácia bodu/úlohy/príkazu**
    - Implementácia - vzlet a pristátie - **povinné**
    - Implementácia - polomer bodu (presnosť regulátora pre daný bod - Hard/Soft) - **povinné**
    - Implementácia - riadenie yaw v špecifickom uhle - **povinné**
6. **Dokumentácia**

- Analýza každého použitého prístupu - **povinné**
  - Výhody a nevýhody
  - Vysvetlenie algoritmov alebo implementácií
- Celkový diagram riešenia - **povinné**
  - Cesty spracovania dát
  - Diagram riadenia dronu v ROS

2 Implementácia

Implementácia projektu zahŕňa niekoľko kľúčových komponentov vrátane načítania máp, vyhľadávania ciest, optimalizácie trajektórie a vytvárania ROS uzlov pre publikovanie príkazov a sledovanie pozície dronu.

2.1 Načítanie máp a príprava prekážok

V projekte sme si vygenerovali názvy súborov máp, ktoré následne načítavame do triedy `nav_msgs::msg::OccupancyGrid` z balíčka Navigation2 v ROS2. Po načítaní máp rozširujeme prekážky o 4 pixely, čím zabezpečujeme, že dron bude mať dostatočný odstup od prekážok.

2.2 Plánovanie misie a generovanie trajektórie

Po načítaní máp načítame misiu, ktorá definuje sekvenciu bodov, ktorými musí dron prejsť. Pre generovanie cesty používame algoritmus A*, ktorý je vhodný na nájdenie optimálnej cesty v mriežke. Algoritmus A* kombinuje hľadanie najkratšej vzdialenosti s heuristikou vzdialenosti od cieľa, čo mu umožňuje efektívne nájsť cestu.

Po vygenerovaní cesty eliminujeme nadbytočné body a ponecháme iba zlomové body, čo zvyšuje efektivitu pohybu. Pri pohybe po diagonále navyše odstraňujeme veľký počet bodov. Tento prístup je opísaný nižšie:

```
double dx1 = curr.x - prev.x;
double dy1 = curr.y - prev.y;
double dx2 = next.x - curr.x;
double dy2 = next.y - curr.y;

bool is_approx_straight = !(dx1 * dy2 != dy1 * dx2);
bool is_approx_diagonal = (std::abs(dx1) > tolerance && std::abs(dy1) > tolerance) &&
                    (std::abs(dx1 - dx2) <= tolerance && std::abs(dy1 - dy2) <= tolerance);

if ((!is_approx_diagonal && !is_approx_straight) ||
   (is_approx_straight && (dx1 * dx2 < 0 || dy1 * dy2 < 0))) // Zmena smeru
{
   optimized_path.push_back(path_points[i]);
}
```

Výpis 1: Optimalizácia cesty - odstránenie bodov

2.3 Funkcia pre riadenie pohybu: `go_to_point`

Vytvorili sme funkciu `void go_to_point(double x, double y, double z, double tolerance)`, ktorá prijíma súradnice cieľa po transponovaní, pretože mapy, simulácia a dron majú rôzne orientácie osí. Parameter `tolerance` určuje presnosť, s akou má dron

doraziť na cieľové súradnice:

- Pre **soft** presnosť je tolerancia 0.1 m.
- Pre **hard** presnosť je tolerancia 0.05 m.
- Pre prechodové body je tolerancia 0.25 m.

Príkazy pre pohyb dronu publikujeme s frekvenciou 10 Hz. Funkcia zároveň kontroluje, či je aktuálna poloha dronu v rámci požadovanej tolerancie. Ak áno, nastaví sa nový cieľový bod.

3 Realizácia

3.1 Vyhľadávanie ciest

Na plánovanie trajektórie dronu sa používajú algoritmy na vyhľadávanie ciest, ako sú A* alebo RRT. Tieto algoritmy zabezpečujú, že dron môže navigovať efektívne medzi bodmi v mape, pričom sa vyhýba prekážkam.

3.1.1 Odstraňovanie bodov

Po vygenerovaní počiatočnej trajektórie sa vykoná optimalizácia odstránením nadbytočných bodov. Tento proces zahŕňa elimináciu sekvencií bodov, ktoré sú v priamej línii (horizontálnej, vertikálnej alebo diagonálnej), čím sa minimalizuje počet bodov a zvyšuje efektivita pohybu dronu.

3.2 Vytváranie Publishera

ROS uzol vytvára Publisher, ktorý publikuje príkazy pre pohyb dronu na základe bodov trajektórie. Publisher je navrhnutý tak, aby odosielal pozíciu dronu na ďalší bod, kým dron nedosiahne cieľovú pozíciu alebo kým nenastane potreba zastaviť publikovanie.

3.2.1 Kontrola skutočnej polohy pre zastavenie publikovania

Počas publikovania príkazov na pohyb dronu uzol pravidelne kontroluje skutočnú polohu dronu. Ak dron dosiahne cieľový bod alebo je v blízkosti s dostatočnou presnosťou, publikovanie sa zastaví. Táto kontrola je dôležitá na minimalizovanie počtu odoslaných príkazov a na zabezpečenie presnosti pohybu dronu v rámci požadovaných súradníc.

3.3 Funkcie pre nastavenie režimu a armovanie dronu

3.3.1 `set_arm`

Táto funkcia zaisťuje armovanie dronu. Funkcia vytvára požiadavku na armovanie dronu pomocou ROS služby a čaká na potvrdenie úspešného armovania. Ak armovanie prebehne úspešne, funkcia pokračuje. Ak sa vyskytnú problémy, vygeneruje sa chybové hlásenie. Kód pre funkciu je nasledovný:

```
void set_arm(){
   auto arm_request = std::make_shared<mavros_msgs::srv::CommandBool::Request>();
   arm_request->value = true;

   while (!arming_client_->wait_for_service(1s))
   {
      if (!rclcpp::ok())
      {
         RCLCPP_ERROR(this->get_logger(), "Interrupted while waiting for the set_mode
             service. Exiting.");
```

```
            return;
        }
        RCLCPP_INFO(this->get_logger(), "Waiting for set_mode service...");
    }

    auto arm_result = arming_client_->async_send_request(arm_request);
    if (rclcpp::spin_until_future_complete(this->get_node_base_interface(), arm_result) ==
        rclcpp::FutureReturnCode::SUCCESS)
    {
        if (arm_result.get()->success)
        {
            RCLCPP_INFO(this->get_logger(), "Drone armed successfully.");
            while (!current_state_.armed)
            {
                rclcpp::spin_some(this->get_node_base_interface());
                std::this_thread::sleep_for(std::chrono::milliseconds(100));
            }
        }
        else
        {
            RCLCPP_ERROR(this->get_logger(), "Failed to arm the drone.");
            return;
        }
    }
}
```

Výpis 2: Funkcia pre armovanie dronu

3.3.2 set_mode

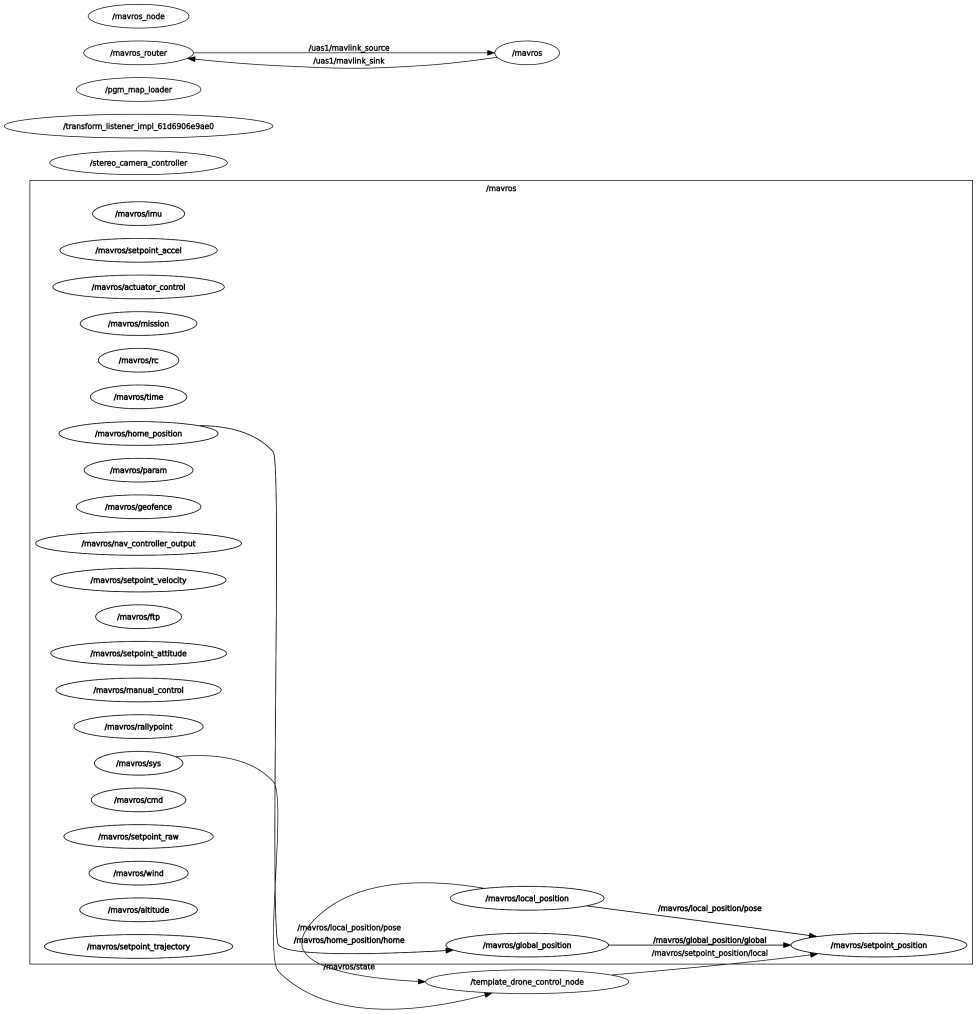
Funkcia `set_mode` nastavuje režim dronu, napríklad na `GUIDED` alebo `LAND`. Funkcia posiela požiadavku na nastavenie režimu a overuje, či bol režim úspešne zmenený. Ak je režim nastavený na `LAND`, funkcia čaká, kým dron nedosiahne výšku 0.15 m pred ukončením. Kód pre funkciu je nasledovný:

```
void set_mode(char *mode){
    mavros_msgs::srv::SetMode::Request guided_set_mode_req;
    guided_set_mode_req.custom_mode = mode;
    while (!set_mode_client_->wait_for_service(1s))
    {
        if (!rclcpp::ok())
        {
            RCLCPP_ERROR(this->get_logger(), "Interrupted while waiting for the set_mode
                service. Exiting.");
            return;
        }
        RCLCPP_INFO(this->get_logger(), "Waiting for set_mode service...");
    }
    auto result = set_mode_client_->async_send_request(std::make_shared<
                    mavros_msgs::srv::SetMode::Request>(guided_set_mode_req));
    if (rclcpp::spin_until_future_complete(this->get_node_base_interface(), result) ==
        rclcpp::FutureReturnCode::SUCCESS)
```

```
    {
        RCLCPP_INFO(this->get_logger(), "Drone mode changed to %s", mode);
    }
    else
    {
        RCLCPP_ERROR(this->get_logger(), "Failed to change mode to %s", mode);
        return;
    }
    if(mode == "LAND"){
        while(current_local_pos_.pose.position.z > 0.15){
            rclcpp::spin_some(this->get_node_base_interface());
            std::this_thread::sleep_for(std::chrono::milliseconds(100));
        }
    }
}
```

Výpis 3: Funkcia pre nastavenie režimu dronu

3.4 Node graf



Obr. 3.1: Node graf

Záver

V tomto projekte sme úspešne vytvorili autonómny systém na riadenie dronu s využitím ROS2. Podarilo sa nám implementovať kľúčové komponenty vrátane načítania máp a ich spracovania, algoritmov na vyhľadávanie ciest a optimalizáciu trajektórie, ako aj vytvorenie ROS uzlov pre komunikáciu a kontrolu nad dronom.

Načítanie a inflácia prekážok boli realizované efektívne, čím sme zabezpečili bezpečný pohyb dronu v zložitom prostredí. Plánovanie trajektórie pomocou algoritmu A* umožnilo efektívne a presné navigovanie medzi bodmi, pričom boli odstránené nadbytočné body, čím sa zlepšila efektivita letu. Funkcie `set_arm` a `set_mode` poskytli spoľahlivý spôsob na armovanie dronu a zmenu jeho režimu, čo je dôležité pre bezpečnosť a flexibilitu pri vykonávaní úloh.

Celkovo sme dosiahli funkčný a spoľahlivý autonómny navigačný systém, ktorý môže byť základom pre ďalšie vylepšenia a rozšírenia, ako je integrácia pokročilých senzorov alebo optimalizácia algoritmov pre ešte efektívnejšie vyhľadávanie ciest. Tento projekt predstavuje významný krok vpred v oblasti autonómnej robotiky a môže nájsť využitie v rôznych aplikáciách, kde je potrebná autonómna navigácia v reálnom čase.