Valsts Pētījumu Programma “COVID-19 seku mazināšanai”

Projekts “Drošu tehnoloģiju integrācija aizsardzībai pret Covid-19 veselības aprūpes un augsta riska zonās”

Nodevums Nr. 4.3_1

Mobilā dezinfekcijas robota kontroles sistēmas izstrāde

Rīga,
Decembris, 2020

Ziņas struktūra ko nosuta datorredzes iekārta:

```
struct MsgEdiCamSectors
{
    char eventBody[254];
};
```

Ziņas saturam papildus tiek pievienota informācija par tas mērķi un ziņas saturam izrēķināta kontrolsumma. Ziņas saturs ir strukturēts JSON formātā:

```
{
  "ver":0.1,
  "data" : {
    "sector_1" : false,
    "sector_2" : true,
    "sector_3" : false
  }
}
```

Lēmumu pieņemšanas programmatūra:

Strukturēta ziņa tiek saņemta no attēlu apstrādes iekārtas caur UDP protokolu un ievietota ziņu rindā.

```
namespace zbot {
    // creates slave session for slamd
    EdiCamSession::EdiCamSession(int socket, MessageQueue<MsgEdiCamSectors>* queue)
        : BaseSession(socket, zbot::SessionType::Response), ediCamQueue(queue) {}

    void EdiCamSession::ProcessMessage(zbot::ContentType type, void* data) {
        syslog(LOG_INFO, "Got EDI data!");
        switch (type) {
        case ContentTypeEdiCamSectors:
        {
            syslog(LOG_INFO, "Got ContentTypeEdiCamSectors!");
            MsgEdiCamSectors* msg = (MsgEdiCamSectors*)data;
            ediCamQueue->PushData(msg);
            break;
        }
        default: syslog(LOG_ERR, "Undefined content type %d", type);
        }
    }
}
```

Ziņa tiek atšifrēta un izgūta informācija par to vai kādā no sektoriem tiek pamanīts smidzināmais objekts.

```
while (ediCamQueue->IsPending()) {
    MsgEdiCamSectors msgEdiCamSectors = *ediCamQueue->GetOldest();
    Json::Reader reader;
    Json::Value root;

    bool parsingSuccessful = reader.parse(msgEdiCamSectors.eventBody, root);
    if (!parsingSuccessful)
    {
        syslog(LOG_INFO, "Failed to parse JSON!");
    }
    else
    {
        Json::Value sectors = root["sector_1"];
        if(sectors.asInt() == 1)enviroment->ediCamSector = 1;
        sectors = root["sector_2"];
        if (sectors.asInt() == 1)enviroment->ediCamSector = 2;
        sectors = root["sector_3"];
        if (sectors.asInt() == 1)enviroment->ediCamSector = 3;
        enviroment->ediCamTimeout = Helper::GetTimeStamp() + enviroment->ediCamTimeoutDelay;
        syslog(LOG_INFO, "Got sector: %d", enviroment->ediCamSector);
    }

}
```

Ziņa par sektoru kurā atrodas objekts tiek nosūtītā uz iekārtu vadības programmatūru.

```
void ReactdSession::SendClean(EnvState * env) {
    MsgDeviceInstruction *msg = new MsgDeviceInstruction();
    msg->mode = DeviceInstructionClean;
    msg->data0 = env->cleanEnable;
    msg->data1 = env->ediCamSector; // Desinfectant sector
    msg->data2 = 0;
    msg->data3 = 0;
    Send(zbot::ContentTypeDeviceInstruction, msg);
    delete msg;
}
```

Atkarībā no tā, kurā segmentā atrodas smidzināmais objekts, robota ātrums tiek mainīts lai panāktu vajadzīgo dezinfekcijas līdzekļa smidzināšanas daudzumu.

```
std::vector<double> driveValues = enviroment->InverseKinematics(*activeTrajectory->GetX(enviroment->pointListState),
                                                                *activeTrajectory->GetY(enviroment->pointListState));
if(enviroment->ediCamSector == 1)enviroment->DifDrive(driveValues[1] * *speed * 0.3, driveValues[0] * *speed * 0.3);
else enviroment->DifDrive(driveValues[1] * *speed, driveValues[0] * *speed);
```

Iekārtu vadības programmatūra:

No lēmumu pieņemšanas programmatūras tiek saņemta strukturēta ziņa ar izpildāmo komandu. Atkarībā no tā, kurā segmentā atrodas objekts, vajadzīgas sprauslas tiek ieslēgtas vai izslēgtas.

```
void ControlSession::DeviceInstruction(void *data) {
    MsgDeviceInstruction *msg = (MsgDeviceInstruction*)data;
    switch (msg->mode) {
```

```
case DeviceInstructionClean:
    switch (msg->data1)
    {
    case 1:
        KernelProcess::instance->CovTurnOneOn();
        break;
    case 2:
        KernelProcess::instance->CovTurnOneOff();
        //syslog(LOG_INFO, "Started sprinkle");
        break;
    case 3:
        KernelProcess::instance->CovTurnOneOff();
        break;
    default:
        KernelProcess::instance->CovTurnOneOff();
        break;
    }
    break;
```

```
void KernelProcess::CovTurnOneOn()
{
    CovTurnOn(DEF_CHOSEN_ONE);
}

void KernelProcess::CovTurnOneOff()
{
    CovTurnOff(DEF_CHOSEN_ONE);
}
```

```
void KernelProcess::CovTurnOn(unsigned char arg)
{
    covidController->TurnOnSprinkler(arg);
}

void KernelProcess::CovTurnOff(unsigned char arg)
{
    covidController->TurnOffSprinkler(arg);
}
```

Ziņa par sprauslas ieslēgšanu/izslēgšanu tiek nosūtīta uz mikrokontrolieri.

```
void CovidController::TurnOnSprinkler(unsigned char arg)
{
    SendCommand(5, arg);
}
void CovidController::TurnOffSprinkler(unsigned char arg)
{
    SendCommand(6, arg);
}
void CovidController::SetSprinklersLeft(unsigned char arg)
{
    SendCommand(7, arg);
}

void CovidController::SetSprinklersRight(unsigned char arg)
{
    SendCommand(8, arg);
}

void CovidController::ReadCurrentState()
{
    SendCommand(9, 2);
}
```

Lai izvairītos no nekorektu ziņu sūtīšanās tiek izrēķināta kontrolsumma.

```
    }

    bool CovidController::SendCommand(unsigned char command, unsigned char argument)
    {
        char buff[4];
        buff[0] = command + '0';
        buff[1] = argument + '0';
        buff[2] = buff[0] + buff[1];
        buff[3] = '\r';
#ifdef DEBUG_COVID
        printf("Command sent: %3s\n", buff);
#endif // DEBUG_COVID
        return WriteData(buff, 4);
    }
```

```
}

//! Make sure that you are also sending a carriage return (hex 0d) at the end of your string.
bool SerialPort2::WriteData(char *data, int len) {
    int len1 = write(fd, data, len);
    if (len1 != len) {
        syslog(LOG_ERR, "Write IO error: %s, len=%d -> %d", strerror(errno), len, len1);
        return false;
    }
    /*for (int i = 0; i < len; i++) printf("%hhX ", data[i]);
    printf("\r");*/
    return true;
}
```

Mikrokontrolieris:

Saņemot ziņu no robota, ziņa tiek pārbaudītā un tiek izgūta vadības komanda un vadāmas sprauslas numurs.

```
void loop() {
  if(Serial.available()){
    //read serial
    c = Serial.read();
    message[ind++] = c;
  }
  //check if message is complete
  if(c == '\r'){
    if(ind >= 4){
      if(message[ind-2] == message[ind-4] + message[ind-3] && message[ind-1]=='\r'){
        cmd =  message[ind-4];
        arg =  message[ind-3]-'0';
```

Attiecīga sprausla ir ieslēgta/izslēgta un tiek nosūtīta atbilde uz robotu par attiecīgas komandas izpildīšanu.

```
switch(cmd){
  case '5':
  //Turn on relays
  TurnOnRelay(arg);
  Serial.print('5');
  Serial.print('\r');
  Serial.print('\n');
  break;
  case '6':
  //Turn off relays
  TurnOffRelay(arg);
  Serial.print('6');
  Serial.print('\r');
  Serial.print('\n');
  break;
  case '7':
  //Set left side
  SetRelays(arg&&0b11111);
  Serial.print('7');
  Serial.print('\r');
  Serial.print('\n');
  break;
  case '8':
  //Set right side
  SetRelays(arg<<5);
  Serial.print('8');
  Serial.print('\r');
  Serial.print('\n');
  break;
  case '9':
  //Set right side
  Serial.print(current_state);
  Serial.print('\r');
  Serial.print('\n');
  break;
}
```

Sprauslas kontrole notiek ar attiecīgo releju ieslēgšanu uz dezinfekcijas līdzekļa smidzināšanas iekārtas, padodot 5V signālu.

Kontaktpersona par tehniskiem jautājumiem: Prof. Agris Ņikitenko (agris.nikitenko@rtu.lv)