ROS

Sumário

1 Introduçao

1.1 Elementos Básicos

- Nós (nodes): executaveis, processos, códigos que realizam alguma atividade, não necessariamente no mesmo sistema
- Tópicos (topics): forma de troca de informações entre os Nós
- Serviços (services): interação entre nós de forma a solicitar uma ação ou um dado como resposta
- Pacotes (packages): forma de organização dos arquivos e codigos no Ros.favorece a modularidade das aplicações do sistema.

1.2 Links uteis

Instalação completa framework site-passo-a-passo site-video
Documentacao acerca do framework wiki ros

Documentacao do que já foi feito no ROS Tutorais ros

Forum para tirar duvidas. ROS Answer

discursões e divulgações do que será feito Discourse ROS

2 Instalação

2.1 Passos da Instalação

video de instalação
link para os passos na wiki ROS

3 Primeiro projeto

Antes de criar um novo pacote, é importante garantir que o ambiente ROS esteja devidamente configurado. Isso é feito utilizando o comando source para carregar as variáveis de ambiente do ROS.
Utilize o comando $catkin_create_pkg$ para criar um novo pacote. O comando abaixo cria um pacote chamado $nome_project$ com dependências nas bibliotecas roscpp, rospy e $std_msgs:$

Comandos no terminal bash linux

```

    source devel/setup.bash // sempre manter atualizado
    catkin_create_pkg nome_project roscpp rospy std_msgs /criando um package
    roscpp <diretorio_destino>
```

- CMakeLists.txt
- package.xml
- include/
- *nome_project/*
- src/, msg/

4 Nós e topicos

Podem ser instaciados multiplas vezes

5 primeiro Nó

talker_cpp.cpp

```
        catkinmake
        rostopic list
        rostopic echo /chatter

        O codigo esta usando o publisher rostopic echo /chatter
        esse comando mostra o que esta acontecendo dentro do publisher

        rostopic hz /chatter // mostra a frequencia de amostragem

        rostopic info /chatter quem ta publicando e quem ta escrito no no
```

talker_cpp.cpp

```
        #include "ros/ros.h"
        #include "std_msgs/String.h"


        int main (int args, char **argv){
            ros::init(args, argv, "talker_cpp");

            ros::NodeHandle nh;

```

```
        ros::Rate loop_rate(10);

        ros:: Publisher chatter_pub = nh.advertise<std_msgs::String>("chatter", 1000);

        int count = 0;
        while(ros:: ok()){
            std::string txt = "Ola ross! contagem:" + std::to_string(count);

            ROS_INFO("%s", txt.c_str());

            std_msgs::String msg;
            msg.data = txt;
            chatter_pub.publish(msg);

            //fazer algo
            ros::spinOnce();
            loop_rate.sleep();
            ++count;
        }

        return 0;
    }
```

5.1 codigo detalhado

headers

```
        #include "ros/ros.h"
        #include "std_msgs/String.h"
```

ros/ros.h : Inclui as funcionalidades basicas do ROS, como a inicializacao, criação de nos, e controle de taxa.
std_msgs/String.h: Inclui a definição da mensagem do tipo String, que é usada para enviar textos.

Função main

```
        int main (int args, char **argv){
        ros::init(args, argv, "talker_cpp");
```

ros::init(args, argv, "talker_cpp");: Inicializa o nó ROS com o nome "talker_cpp".

Criação de um NodeHandle

```
        ros::NodeHandle nh;
```

Cria um manipulador de nó, responsável por interagir com o sistema ROS (criando tópicos, assinantes, etc.).

Definição da Taxa de Loop

```
        ros::Rate loop_rate(10);
```

ros::Rate loop_rate(10);: Define a taxa de repetição do loop para 10 Hz.

Criando o Publicador

```
        ros::Publisher chatter_pub = nh.advertise<std_msgs::String>("chatter", 1000);
```

Cria um publicador no tópico chatter para mensagens do tipo std_msgs::String. O segundo argumento (1000) é o tamanho da fila de mensagens.

5.2 explicação terminal

No exemplo feito em aula, ele rodava quatro terminais ao mesmo tempo, um com o roscore, rosrun my_first_pkg/
ele não acha as bibliotecas do ros e a biblioteca standart mensage , porém o projeto funciona perfeitamente
cmakelists.txt
Defina o executável talker_cpp e seu arquivo fonte

cmakeLists.txt

```

        add_executable(talker_cpp src/talker_cpp.cpp)}\\

        # Vincule as bibliotecas necessarias ao executavel||
        target_link_libraries(talker_cpp \${catkin_LIBRARIES})\\
```

6 Nó em python

o Primeiro comando é importante pois ele seta o compilador que vai ser usado

base de um nó em python

```
    #!/usr/bin/env python3

    import rospy


    def talker():
        rospy.init_node("talker_python")

        loop_rate = rospy.Rate(10)
```

```
        while not rospy.is_shotdown():
            loop_rate.sleep()


    if __name__ =="__main__":
        try:
            talker()
        except rospy.ROSInterruptException:
            pass
```

6.1 detalhameento do código

Codigo

```
#!/usr/bin/env python3
```

- Shebang (#): Informa ao sistema operacional que este script deve ser executado usando o interpretador Python 3. O caminho (/usr/bin/env python3) localiza automaticamente o Python 3 no sistema, garantindo que o script seja executado corretamente, independentemente da localizacao do Python 3.

importacao

```
import rospy
```

- Importação do módulo rospy: O módulo rospy é a biblioteca ROS para Python, que permite criar nós ROS, publicar/-subscrever em tópicos, usar serviços, etc.

Função talker

```
def talker():
rospy.init_node("talker_python")
```

- Define uma função chamada talker.
- rospy.init_node("talker_python"): Inicializa um nó ROS chamado "talker_python". Todo nó ROS precisa de um nome exclusivo, e esta linha registra o nó no mestre ROS.

loop principal

```
loop_rate = rospy.Rate(10)
```

- rospy.Rate(10): Define uma taxa de loop de 10 Hz, ou seja, o loop rodará 10 vezes por segundo. Essa função é usada para controlar a frequência de execução do loop principal.

base de um nó em python

```
while not rospy.is_shutdown():
loop_rate.sleep()
```

- Loop while: Este loop while continuará rodando até que o ROS seja desligado (por exemplo, ao pressionar Ctrl+C).
- rospy.is_shutdown(): Verifica se o ROS foi desligado.
- loop_rate.sleep(): Faz o loop dormir o tempo necessário para manter a frequência definida bem rospy.Rate(10). Neste caso, ele dorme por 0,1 segundos para manter a taxa de 10 Hz.

diretiva main

```
if __name__ == "__main__":
try:
    talker()
except rospy.ROSInterruptException:
    pass
```

- if __name__ == "__main__":: Este bloco verifica se o script está sendo executado diretamente (não importado como um módulo). Se for o caso, ele executa a função talker.
- try-except: O bloco try tenta executar a função talker(). Se uma excecao rospy.ROSInterruptException for lançada (geralmente ocorre quando o ROS e interrompido), o bloco except captura essa excecao e ignora-a (o código dentro de pass não faz nada).

7 ROS Publisher e Subscriber em C++ - ParteI

terminal ROS

```
rostopic -h
rostopic hz <nome_topico> mostra a publicacao do topico com a frequencia estabelecida no
    codigo
rosnode list lista os nos disponiveis
rostpic list // deve mostrar os topicos em execucao
rostopic pub /number std_msgs/Float64 "data: 4.0"
rostopic pub /number std_msgs/Float64 "data: 4.0 " -r 2 // iteracao sobre o comando na
                    taxa de 2 hz.
 rostopic -h //lista de comandos associados ao rostopic
 rostopic info <nome_topico>
envia informacoes para o topico en execucao
```

counter.cpp

```cpp
#include <iostream>
#include "ros/ros.h"
#include"std_msgs/Float64.h"
class Counter{
    public:
        Counter(ros::NodeHandle *nh){
            count = 0;

            num_sub = nh ->subscribe("/number", 10, &Counter::numberCallBack, this);
        }
        void numberCallBack(const std_msgs::Float64 &msg){
            count = count + msg.data;
            ROS_INFO("CONTAGEM ATUAL:%f", count);
        }
    private:
        double count;
        ros::Subscriber num_sub;
};

int main(int argc, char **argv){
    ros::init(argc, argv, "counter_node");
    ros:: NodeHandle nh;
    Counter counter = Counter( &nh);
    ros::spin();

    return 0;
```

7.1 detalhamento counter.cpp

- #include "ros/ros.h": Inclui a biblioteca principal do ROS, que fornece as funcionalidades necessárias para criar nós, publicar e subscrever tópicos, etc.
- #include "std_msgs/Float64.h": Inclui a definição do tipo de mensagem std_msgs::Float64, que é usada para enviar dados do tipo double em ROS.

declaração da classe counter

```cpp
class Counter {
    public:
        Counter(ros::NodeHandle *nh) {
            count = 0;
            num_sub = nh->subscribe("/number", 10, &Counter::numberCallBack, this
                );
        }

        void numberCallBack(const std_msgs::Float64 &msg) {
            count = count + msg.data;
            ROS_INFO("CONTAGEM ATUAL: %f", count);
        }

    private:
        double count;
        ros::Subscriber num_sub;
};
```

- Classe Counter: Define uma classe que gerencia a contagem dos valores recebidos e armazena o estado necessário.
- Construtor Counter(ros::NodeHandle *nh):
  - Este é o construtor da classe, que é chamado quando uma instância da classe Counter é criada.
  - count = 0;: Inicializa a variável count com 0
  - num_sub = nh subscribe("/number", 10, &Counter::numberCallBack, this);:
  - Cria um subscritor (Subscriber) que se inscreve no tópico /number.
  - 10: O segundo argumento é o tamanho da fila de mensagens.
  - &Counter::numberCallBack: Especifica o método que será chamado quando uma nova mensagem for recebida (callback).
  - this: Passa um ponteiro para a instância atual da classe Counter para o método de callback.
- Método numberCallBack
  - void numberCallBack(const std_msgs::Float64 &msg): Método que será chamado sempre que uma nova mensagem for recebida no tópico /number.
  - count = count + msg.data;: Atualiza a variável count somando o valor recebido (msg.data).
  - OS_INFO("CONTAGEM ATUAL: %f", count);: Imprime a contagem atual no terminal, formatando o número com

Função main

```cpp
 int main(int argc, char **argv) {
ros::init(argc, argv, "counter_node");
ros::NodeHandle nh;
Counter counter = Counter(&nh);
ros::spin();
return 0;
}
```

- int main(int argc, char **argv): A função principal que inicia o programa.
- ros::init(argc, argv, "counter_node");: Inicializa o nó ROS com o nome "counter_node".
- ros::NodeHandle nh;: Cria um objeto NodeHandle, que é necessário para interagir com o sistema ROS.
- Counter counter = Counter(&nh);: Cria uma instância da classe Counter, passando o NodeHandle para que o subscritor possa ser criado.
- ros::spin();: Entra em um loop que mantém o programa rodando e processando callbacks até que o ROS seja desligado.
- return 0;: Retorna 0 ao final da execução do programa, indicando que o programa terminou com sucesso.

7.2 definindo um timer no counter.cpp

counter.cpp - publicando

```
#include <iostream>
#include "ros/ros.h"
#include"std_msgs/Float64.h"

class Counter{
    public:
        Counter(ros::NodeHandle *nh){
            count = 0;
            publish_interval = 1;

            num_sub = nh ->subscribe("/number", 10, &Counter::numberCallBack, this);
            count_pub = nh -> advertise < std_msgs::Float64>("/current_count", 10);
            timer_pub = nh -> createTimer(ros::Duration(publish_interval), &Counter::
                timerCallback, this);

        }
        void numberCallBack(const std_msgs::Float64 &msg){
            count = count + msg.data;
            ROS_INFO("CONTAGEM ATUAL:%f", count);


        }

        void timerCallback(const ros::TimerEvent &event){
            std_msgs ::Float64 msg;
            msg.data = count;
            count_pub.publish(msg);

        }


    private:
        double count;
        double publish_interval;
        ros::Subscriber num_sub;
        ros:: Publisher count_pub;
        ros:: Timer timer_pub;
};

int main(int argc, char **argv){
    ros::init(argc, argv, "counter_node");
    ros:: NodeHandle nh;
    Counter counter = Counter( &nh);
    ros::spin();

    return 0;

}
```

7.2.1 detalhamento do timercallback

- publish_interval: Essa variável foi adicionada à classe Counter e é usada para definir o intervalo de tempo (em segundos) entre cada publicação da contagem atualizada. No construtor, ela é inicializada com o valor 1, o que significa que a contagem será publicada a cada segundo.

Criação do Publisher count_pub

```
            count_pub = nh->advertise<std_msgs::Float64>("/current_count", 10)
```

- count_pub: Este é um publisher que foi adicionado à classe Counter. Ele publica mensagens do tipo std_msgs::Float64 no tópico /current_count.
- /current_count: Esse é o novo tópico onde a contagem atualizada será publicada. Outros nós podem se inscrever nesse tópico para receber as contagens.
- 10: Tamanho da fila de mensagens, semelhante ao subscritor.

Criação do Publisher count_pub

```
            timer_pub = nh->createTimer(ros::Duration(publish_interval), &Counter::
                timerCallback, this);
```

- timer_pub: Um temporizador foi adicionado à classe Counter. Esse temporizador é criado com um intervalo definido por publish_interval (inicialmente 1 segundo) e chama o método timerCallback a cada intervalo.
- ros::Duration(publish_interval): Define o intervalo do temporizador com base na duração especificada em publish_interval.

Criação do Publisher count_pub

```
            void timerCallback(const ros::TimerEvent &event) {
                std_msgs::Float64 msg;
                msg.data = count;
                count_pub.publish(msg);
            }
```

- timerCallback: Este método é chamado automaticamente a cada intervalo de tempo definido pelo temporizador.
- msg.data = count;: Cria uma mensagem do tipo std_msgs::Float64 e define seu valor como o valor atual de count.
- ount_pub.publish(msg);: Publica a mensagem no tópico /current_count.

8 Publisher e subcriber em c++ partII

number_publish.cpp

```
#include "ros/ros.h"
#include "std_msgs/Float64.h"

class NumberPublisher{
    public:
        NumberPublisher(ros::NodeHandle *nh){
            number = 1.2;
            publish_interval = 1;

            num_pub = nh->advertise <std_msgs::Float64>("/number", 10);
            timer_pub = nh -> createTimer (ros::Duration(publish_interval), &NumberPublisher::
                timerCallback, this);
        }

        void timerCallback(const ros:: TimerEvent &event){
            std_msgs::Float64 msg;
            msg.data = number;
            num_pub.publish(msg);
        }

    private:
        double number;
        double publish_interval;
        ros::Publisher num_pub;
        ros::Timer timer_pub;
};


int main(int argc, char **argv)
{
    ros::init(argc, argv, "number_publisher");
    ros::NodeHandle nh;
    NumberPublisher num_publisher = NumberPublisher(&nh);

    ros::spin();
    return 0;

}
```

minimal_node_cpp

```
#include "ros/ros.h"


class NodeName{                             // Atualizar

    public:
        NodeName(ros::NodeHandle *nh){      // Atualizar
            //Inicializar Variaveis

        }

    private:

};


int main(int argc, char **argv)
{
    ros::init(argc, argv, "node_name");     // Atualizar
    ros::NodeHandle nh;
    NodeName node_name = NodeName(&nh);     // Atualizar

    ros::spin();
    return 0;
}
```

9 Publisher e subcriber em python part I

counter.py

```python
#!/usr/bin/python3
import rospy

from std_msgs.msg import Float64

class Counter:
    def __init__(self) -> None:
        self.count  = 0
        self.num_sub = rospy.Subscriber("/number", Float64, self.numberCallback, queue_size = 10)


    def numberCallback(self, msg):
        self.count = self.count + msg.data
        rospy.loginfo("contagem atual" + str(self.count))


if __name__ == "__main__":
    try:
        rospy.init_node("counter_node")
        Counter()
        rospy.spin()
    except rospy.ROSInterruptException:
        pass
```

Ros terminal

```
permissao .py
chmod +x <codigo_py>.py
rostopic pub -r 1 /number std_msgs/Float64 "2.0"
roscore //comunicacao com o master
```

counter.py- with publish

```python
#!/usr/bin/python3
import rospy

from std_msgs.msg import Float64

class Counter:
    def __init__(self) -> None:
        self.count  = 0
        self.publish_interval = 3
        self.num_sub = rospy.Subscriber("/number", Float64, self.numberCallback, queue_size = 10)
        self.count_pub = rospy.Publisher("Current_count", Float64, queue_size= 10)
        self.timer_pub = rospy.Timer(rospy.Duration(), self.TimerCallback)

    def numberCallback(self, msg):
        self.count = self.count + msg.data
        rospy.loginfo("contagem atual" + str(self.count))
    def TimerCallback(self, event):
        msg = Float64()
        msg.data = self.count
        self.count_pub.publish(msg)
if __name__ == "__main__":
    try:
        rospy.init_node("counter_node")
        Counter()
        rospy.spin()
    except rospy.ROSInterruptException:
        pass
```

10 publish e subscriber em python partII

number_publish.py

```python
#!/usr/bin/env python3

import rospy
from std_msgs.msg import Float64

class NumberPublisher:                          # Mudar

    def __init__(self):
        self.number = 3.1
        self.publish_interval = 1.5

        self.num_pub = rospy.Publisher("/number", Float64, queue_size = 10)
        self.timer_pub = rospy.Timer(rospy.Duration(self.publish_interval), self.timerCallback)

    def timerCallback(self, event):
        msg = Float64()
        msg.data = self.number
        self.num_pub.publish(msg)
```

```

if __name__ == '__main__':
    try:
        rospy.init_node("number_publisher_node")                # Mudar
        NumberPublisher()
        rospy.spin()
    except rospy.ROSInterruptException:
        pass
```

minimal_pyhton_node

```
#!/usr/bin/env python3

import rospy

class NodeName:                                 # Mudar

    def __init__(self):


if __name__ == '__main__':
    try:
        rospy.init_node(node_name)                  # Mudar
        NodeName()
        rospy.spin()
    except rospy.ROSInterruptException:
        pass
```

11 ROSService server com c++

counter.cpp

```
#include <iostream>
#include "ros/ros.h"
#include"std_msgs/Float64.h"
#include "std_srvs/Empty.h"

class Counter{
    public:
        Counter(ros::NodeHandle *nh){
            count = 0;
            publish_interval = 1;

            num_sub = nh ->subscribe("/number", 10, &Counter::numberCallBack, this);
            count_pub = nh -> advertise < std_msgs::Float64>("/current_count", 10);
            timer_pub = nh -> createTimer(ros::Duration(publish_interval), &Counter::timerCallback
                , this);
            reset_srv = nh -> advertiseService ("/reset_counter",  &Counter::resetServerCallback,
                this);

        }
        // callback do subscribe
        void numberCallBack(const std_msgs::Float64 &msg){
            count = count + msg.data;
            ROS_INFO("CONTAGEM ATUAL:%f", count);


        }
        //callback do publish
        // tempo do ros e nao do computador
        void timerCallback(const ros::TimerEvent &event){
            std_msgs ::Float64 msg;
            msg.data = count;
            count_pub.publish(msg);

        }
        bool resetServerCallback(std_srvs::Empty::Request &req, std_srvs::Empty::Response  &res){
            count = 0;
            ROS_INFO("resetando a contagem");
            return true;
        }


    private:
        double count;
        double publish_interval;
        ros::Subscriber num_sub;
        ros:: Publisher count_pub;
        ros:: Timer timer_pub;
        ros::ServiceServer reset_srv;
};

int main(int argc, char **argv){
    ros::init(argc, argv, "counter_node");
    ros:: NodeHandle nh;
    Counter counter = Counter( &nh);
```

```
    ros::spin();

    return 0;

}
```

number_publisher.cpp

```
#include "ros/ros.h"

#include "std_msgs/Float64.h"

class NumberPublisher{

    public:
        NumberPublisher(ros::NodeHandle *nh){
            number = 1.2;
            publish_interval = 1;

            num_pub = nh->advertise<std_msgs::Float64>("/number", 10);
            timer_pub = nh->createTimer(ros::Duration(publish_interval), &NumberPublisher::
                timerCallback, this);

        }

        void timerCallback(const ros::TimerEvent &event){
            std_msgs::Float64 msg;
            msg.data = number;
            num_pub.publish(msg);
        }

    private:
        double number;
        double publish_interval;
        ros::Publisher num_pub;
        ros::Timer timer_pub;

};


int main(int argc, char **argv)
{
    ros::init(argc, argv, "number_publisher");
    ros::NodeHandle nh;
    NumberPublisher num_publisher = NumberPublisher(&nh);

    ros::spin();
    return 0;
}
```

ROSservice terminal

```
rosservice call /reset_counter "{}"
rosservice list
rosservice info <nome_service>
```

12 ROS service client em c++

number_publisher.cpp

```
#include "ros/ros.h"
#include "std_msgs/Float64.h"
#include "std_srvs/Empty.h"

class NumberPublisher{
    public:
        NumberPublisher(ros::NodeHandle *nh){
            number = 1.2;
            publish_interval = 1;
            reset_interval = 7;

            num_pub = nh->advertise <std_msgs::Float64>("/number", 10);
            timer_pub = nh -> createTimer (ros::Duration(publish_interval), &NumberPublisher::
                timerCallback, this);
            timer_reset = nh->createTimer(ros::Duration(reset_interval), &NumberPublisher::
                timeResetCallback, this);
            client_reset = nh -> serviceClient<std_srvs::Empty>("reset_counter");
        }


           void timerCallback(const ros:: TimerEvent &event){
            std_msgs::Float64 msg;
            msg.data = number;
            num_pub.publish(msg);
        }

        void timeResetCallback(const ros:: TimerEvent &event){
```

```
            std_srvs:: Empty srv;
            ROS_INFO("solicitacao de reset da contagem");
            client_reset.call(srv);

        }
    private:
        double number;
        double publish_interval;
        double reset_interval;
        ros::Publisher num_pub;
        ros::Timer timer_pub;
        ros::ServiceClient client_reset;
        ros::Timer timer_reset;

};


int main(int argc, char **argv)
{
    ros::init(argc, argv, "number_publisher");
    ros::NodeHandle nh;
    NumberPublisher num_publisher = NumberPublisher(&nh);

    ros::spin();
    return 0;

}
```

Listing:

```
roscore
rosrun my_project_cpp counter
rostopic echo /current_count
rosrun my_project_cp
rosrun my_project_cpp number_publisher
```

13 ROS service server em python

counter.py

```
#!/usr/bin/python3
import rospy

from std_msgs.msg import Float64
from std_srvs.srv import Empty, EmptyResponse

class Counter:
    def __init__(self) -> None:
        self.count  = 0
        self.publish_interval = 3
        self.num_sub = rospy.Subscriber("/number", Float64, self.numberCallback, queue_size = 10)
        self.count_pub = rospy.Publisher("/current_count", Float64, queue_size= 10)
        self.timer_pub = rospy.Timer(rospy.Duration(), self.TimerCallback)
        self.reset_srv =rospy.Service("/reset_counter", Empty, self.resetSrvCallback)

    def numberCallback(self, msg):
        self.count = self.count + msg.data
        rospy.loginfo("contagem atual" + str(self.count))

    def TimerCallback(self, event):
        msg = Float64()
        msg.data = self.count
        self.count_pub.publish(msg)

    def resetSrvCallback(self, req):
        self.count = 0
        rospy.loginfo("resetando a contagem")
        return EmptyResponse()

if __name__ == "__main__":
    try:
        rospy.init_node("counter_node")
        Counter()
        rospy.spin()
    except rospy.ROSInterruptException:
        pass
```

number_publish.py

```
#!/usr/bin/env python3

import rospy
from std_msgs.msg import Float64


class NumberPublisher:                          # Mudar

```

```
    def __init__(self):
        self.number = 3.1
        self.publish_interval = 1.5


        self.num_pub = rospy.Publisher("/number", Float64, queue_size = 10)
        self.timer_pub = rospy.Timer(rospy.Duration(self.publish_interval), self.timerCallback)

    def timerCallback(self, event):
        msg = Float64()
        msg.data = self.number
        self.num_pub.publish(msg)


if __name__ == '__main__':
    try:
        rospy.init_node("number_publisher_node")                # Mudar
        NumberPublisher()
        rospy.spin()
    except rospy.ROSInterruptException:
        pass
```

14 ROS Service Client em Python

number_publish.py

```
#!/usr/bin/env python3

import rospy
from std_msgs.msg import Float64
from std_srvs.srv import Empty

class NumberPublisher:                          # Mudar

    def __init__(self):
        self.number = 3.1
        self.publish_interval = 1.5
        self.reset_interval = 8

        self.num_pub = rospy.Publisher("/number", Float64, queue_size = 10)
        self.timer_pub = rospy.Timer(rospy.Duration(self.publish_interval), self.timerCallback)
        self.client_reset =rospy.ServiceProxy("/reset_counter", Empty)
        self.timer_reset = rospy.Timer(rospy.Duration(self.reset_interval), self.
            timerResetCallback)

    def timerCallback(self, event):
        msg = Float64()
        msg.data = self.number
        self.num_pub.publish(msg)
    def timerResetCallback(self, event):
        rospy.loginfo("solicitando reset da contagem")
        self.client_reset()



if __name__ == '__main__':
    try:
        rospy.init_node("number_publisher_node")                # Mudar
        NumberPublisher()
        rospy.spin()
    except rospy.ROSInterruptException:
        pass
```

15 Launch files

Os arquivos de lançamento (launch files) em ROS (Robot Operating System) são usados para iniciar múltiplos nós (nodes) e definir suas configurações de maneira automatizada e conveniente. Esses arquivos são escritos em XML e têm a extensão .launch. Eles permitem que o usuário especifique a estrutura e os parâmetros necessários para a execução de um sistema robótico complexo

counter_cpp.launch

```
<launch>
    <!-- Define um argumento chamado 'number_topic' com um valor padrao '/number'.
         Este argumento pode ser substituido na linha de comando ao iniciar o arquivo de
             lancamento. -->
    <arg name="number_topic" default="/number"/>

    <!-- Inicia um no chamado 'counter_cpp' do pacote 'my_project_cpp', executando o tipo 'counter
        '.
         A saida do no e direcionada para a tela ('screen'). -->
    <node name="counter_cpp" pkg="my_project_cpp" type="counter" output="screen">

        <!-- Redireciona o topico '/current_count' para '/count', alterando a comunicacao entre os
            nos
             sem necessidade de modificar o codigo-fonte do no. -->
        <remap from="/current_count" to="/count"/>
```

```
        <!-- Redireciona o topico '/number' para o valor do argumento 'number_topic'.
             Neste caso, sera '/number' a menos que o argumento seja substituido. -->
        <remap from="/number" to="$(arg number_topic)"/>
    </node>

    <!-- Inicia outro no chamado 'counter_cpp2' do pacote 'my_project_cpp', tambem executando o tipo 'counter'.
         A saida deste no tambem e direcionada para a tela ('screen'). -->
    <node name="counter_cpp2" pkg="my_project_cpp" type="counter" output="screen">
        <!-- Este no nao possui remapeamentos de topicos definidos. -->
    </node>
</launch>
```

my_project_cpp.launch

```
<launch>
    <!-- Define um argumento chamado 'new_counter' com um valor padrao 'True'.
         Este argumento pode ser usado para controlar a inclusao condicional de nos ou grupos. -->
    <arg name="new_counter" value="True"/>

    <!-- Inclui outro arquivo de lancamento localizado no pacote 'my_project_bringup'.
         O arquivo incluido e 'counter_cpp.launch'. -->
    <include file="$(find my_project_bringup)/launch/counter_cpp.launch">

        <!-- Passa um argumento para o arquivo de lancamento incluido.
             Define 'number_topic' com o valor 'new_number' no arquivo incluido. -->
        <arg name="number_topic" value="new_number"/>
    </include>

    <!-- Inicia um no chamado 'num_pub_cpp' do pacote 'my_project_cpp', executando o tipo 'number_publisher'.
         A saida do no e direcionada para a tela ('screen'). -->
    <node name="num_pub_cpp" pkg="my_project_cpp" type="number_publisher" output="screen"/>

    <!-- Agrupa nos condicionalmente baseado no valor do argumento 'new_counter'.
         O grupo so sera incluido se 'new_counter' for verdadeiro ('True'). -->
    <group if="$(arg new_counter)">

        <!-- Inicia um no chamado 'num_pub_cpp_2' do pacote 'my_project_cpp', executando o tipo 'number_publisher'.
             A saida deste no tambem e direcionada para a tela ('screen'). -->
        <node name="num_pub_cpp_2" pkg="my_project_cpp" type="number_publisher" output="screen"/>
    </group>

</launch>
```

16 ROS paramater cpp part I

terminal ROS

```
        rosparam set    set parameter
        rosparam get    get parameter
        rosparam load   load parameters from file
        rosparam dump   dump parameters to file
        rosparam delete delete parameter
        rosparam list   list parameter names
```

para setar o parametro no projeto você vai ter que carregar o package individual se não ele nao vai carregar o parametro

a criação do pkg de launch não necessita de pkgs adicionais depois de feito esse processo o pram ele aparce na lista

terminal ROS

```
catkin_create_pkg my_project_bringup

catkin_make --only-pkg-with-deps my_project_cpp
roslaunch <package_name> <launch_name>
```

counter.cpp

```
#include <iostream>
#include "ros/ros.h"
#include "std_msgs/Float64.h"
#include "std_srvs/Empty.h"

class Counter {
public:
    Counter(ros::NodeHandle *nh) {
        count = 0;
        publish_interval = 1;

        nh->setParam("custom_param", false);
        if(nh->getParam("initial_count", count)){
            ROS_INFO("contagem inicida %f", count);
        }else{
            ROS_INFO("contagem inicial nao iniciadada");
```

```
        }
        nh->param<double>("pub_rate", publish_interval, 1);

        num_sub = nh->subscribe("/number", 10, &Counter::numberCallBack, this);
        count_pub = nh->advertise<std_msgs::Float64>("/current_count", 10);
        timer_pub = nh->createTimer(ros::Duration(publish_interval), &Counter::timerCallback, this
            );
        reset_srv = nh->advertiseService("/reset_counter", &Counter::resetServerCallback, this);
    }

    // callback do subscribe
    void numberCallBack(const std_msgs::Float64 &msg) {
        count += msg.data;
        ROS_INFO("CONTAGEM ATUAL: %f", count);
    }

    // callback do publish
    void timerCallback(const ros::TimerEvent &event) {
        std_msgs::Float64 msg;
        msg.data = count;
        count_pub.publish(msg);
    }

    bool resetServerCallback(std_srvs::Empty::Request &req, std_srvs::Empty::Response &res) {
        count = 0;
        ROS_INFO("resetando a contagem");
        return true;
    }

private:
    double count;
    double publish_interval;
    ros::Subscriber num_sub;
    ros::Publisher count_pub;
    ros::Timer timer_pub;
    ros::ServiceServer reset_srv;
};

int main(int argc, char **argv) {
    ros::init(argc, argv, "counter_node");
    ros::NodeHandle nh;//deixando claro que questamos trabalhando com o namespace privado do ROS
    Counter counter(&nh);
    ros::spin();
    return 0;
}
```

counter_cpp.launch

```
<launch>
    <arg name = "number_topic" default= "/number"/>
    <param  name="initial_count" type = "double" value= "3.1415" />
    <param  name="pub_rate" type = "double" value= "2.5" />
    <node name = "counter_cpp" pkg = "my_project_cpp" type="counter" output="screen" >
        <remap from= "/number" to="$(arg number_topic)" />
    </node>
</launch>
```

16.1 adendos

tive muito problema pois tinha definido o nó para namesapces privados, quando busquei o parametro do launch , não achava de jeito nenhum definindo o parametro para o namesapce privado

counter_cpp.launch

```
<launch>
    <arg name="number_topic" default="/number"/>
    <param name="initial_count" type="double" value="3.1415"/>
    <node name="counter_cpp" pkg="my_project_cpp" type="counter" output="screen">
        <remap from="/number" to="$(arg number_topic)"/>
    </node>
</launch>
```

counter.cpp

```
        #include <iostream>
#include "ros/ros.h"
#include "std_msgs/Float64.h"
#include "std_srvs/Empty.h"

class Counter {
public:
    Counter(ros::NodeHandle *nh) {
        count = 0;
        publish_interval = 1.0;

        nh->setParam("custom_param", false);
        if (nh->getParam("initial_count", count)) {
            ROS_INFO("Contagem inicial: %f", count);
```

```
        } else {
            ROS_INFO("Contagem inicial  n  o  iniciada");
        }

        num_sub = nh->subscribe("/number", 10, &Counter::numberCallBack, this);
        count_pub = nh->advertise<std_msgs::Float64>("/current_count", 10);
        timer_pub = nh->createTimer(ros::Duration(publish_interval), &Counter::timerCallback, this
            );
        reset_srv = nh->advertiseService("/reset_counter", &Counter::resetServerCallback, this);
    }

    // Callback do subscribe
    void numberCallBack(const std_msgs::Float64::ConstPtr& msg) {
        count += msg->data;
        ROS_INFO("CONTAGEM ATUAL: %f", count);
    }

    // Callback do publish
    void timerCallback(const ros::TimerEvent& event) {
        std_msgs::Float64 msg;
        msg.data = count;
        count_pub.publish(msg);
    }

    bool resetServerCallback(std_srvs::Empty::Request &req, std_srvs::Empty::Response &res) {
        count = 0;
        ROS_INFO("Resetando a contagem");
        return true;
    }

private:
    double count;
    double publish_interval;
    ros::Subscriber num_sub;
    ros::Publisher count_pub;
    ros::Timer timer_pub;
    ros::ServiceServer reset_srv;
};

int main(int argc, char **argv) {
    ros::init(argc, argv, "counter_node");
    ros::NodeHandle nh("\~");//definindo namespace padr o como o privado
    Counter counter(&nh);
    ros::spin();
    return 0;
}
```

17 ROS parameter cpp partII

17.1 checagem da existencia do parametro

A checagem acontece dentro da classe do nó , e a declaração dentro do launchfile

counter.cpp

```
#include <iostream>
#include "ros/ros.h"
#include "std_msgs/Float64.h"
#include "std_srvs/Empty.h"

class Counter {
public:
    Counter(ros::NodeHandle *nh) {
        count = 0;
        publish_interval = 1;

        if(nh->getParam("initial_count", count)){
            ROS_INFO("contagem inicida %f", count);
        }else{
            ROS_INFO("contagem inicial nao iniciadada");
        }
        nh->param<double>("pub_rate", publish_interval, 1);
        if(nh->hasParam("pub_rate")||nh->hasParam("initial_count")){
            ROS_WARN("Parametros customizados");
            nh->setParam("custom_param", true);
        }else{
           nh->setParam("custom_param", false);
        }

        num_sub = nh->subscribe("/number", 10, &Counter::numberCallBack, this);
        count_pub = nh->advertise<std_msgs::Float64>("/current_count", 10);
        timer_pub = nh->createTimer(ros::Duration(publish_interval), &Counter::timerCallback, this
            );
        reset_srv = nh->advertiseService("/reset_counter", &Counter::resetServerCallback, this);
    }

    // callback do subscribe
    void numberCallBack(const std_msgs::Float64 &msg) {
```

```
        count += msg.data;
        ROS_INFO("CONTAGEM ATUAL: %f", count);
    }

    // callback do publish
    void timerCallback(const ros::TimerEvent &event) {
        std_msgs::Float64 msg;
        msg.data = count;
        count_pub.publish(msg);
    }

    bool resetServerCallback(std_srvs::Empty::Request &req, std_srvs::Empty::Response &res) {
        count = 0;
        ROS_INFO("resetando a contagem");
        return true;
    }

private:
    double count;
    double publish_interval;
    ros::Subscriber num_sub;
    ros::Publisher count_pub;
    ros::Timer timer_pub;
    ros::ServiceServer reset_srv;
};

int main(int argc, char **argv) {
    ros::init(argc, argv, "counter_node");
    ros::NodeHandle nh;//deixando claro que questamos trabalhando com o namespace privado do ROS
    Counter counter(&nh);
    ros::spin();
    return 0;
}
```

counter_cpp.launch

```
<launch>
    <arg name = "number_topic" default= "/number"/>
    <param  name="initial_count" type = "double" value= "3.1415" />
    <param  name="pub_rate" type = "double" value= "2.5" />
    <node name = "counter_cpp" pkg = "my_project_cpp" type="counter" output="screen" >
        <remap from= "/number" to="$(arg number_topic)" />
    </node>
</launch>
```

number_publisher_cpp.launch

```
<launch>
    <arg name = "number_topic" default= "/number"/>
    <param  name="initial_count" type = "double" value= "3.1415" />
    <param  name="pub_rate" type = "double" value= "2.5" />
    <node name = "counter_cpp" pkg = "my_project_cpp" type="counter" output="screen" >
        <remap from= "/number" to="$(arg number_topic)" />
    </node>
</launch>
```

17.2 deletando parametros

counter.cpp

```
#include <iostream>
#include "ros/ros.h"
#include "std_msgs/Float64.h"
#include "std_srvs/Empty.h"

class Counter {
public:
    Counter(ros::NodeHandle *nh) {
        count = 0;
        publish_interval = 1;

        if(nh->getParam("initial_count", count)){
            ROS_INFO("contagem inicida %f", count);
        }else{
            ROS_INFO("contagem inicial nao iniciadada");
        }
        nh->param<double>("pub_rate", publish_interval, 1);
        if(nh->hasParam("pub_rate")||nh->hasParam("initial_count")){
            ROS_WARN("Parametros customizados");
            nh->setParam("custom_param", true);
        }else{
           nh->setParam("custom_param", false);
        }
        if(nh->hasParam("pub_rate")){
            ROS_INFO("deletando parametro");
            nh->deleteParam("pub_rate");
            nh->deleteParam("initial_count");
```

```
        }

        num_sub = nh->subscribe("/number", 10, &Counter::numberCallBack, this);
        count_pub = nh->advertise<std_msgs::Float64>("/current_count", 10);
        timer_pub = nh->createTimer(ros::Duration(publish_interval), &Counter::timerCallback, this
            );
        reset_srv = nh->advertiseService("/reset_counter", &Counter::resetServerCallback, this);
    }

    // callback do subscribe
    void numberCallBack(const std_msgs::Float64 &msg) {
        count += msg.data;
        ROS_INFO("CONTAGEM ATUAL: %f", count);
    }

    // callback do publish
    void timerCallback(const ros::TimerEvent &event) {
        std_msgs::Float64 msg;
        msg.data = count;
        count_pub.publish(msg);
    }

    bool resetServerCallback(std_srvs::Empty::Request &req, std_srvs::Empty::Response &res) {
        count = 0;
        ROS_INFO("resetando a contagem");
        return true;
    }

private:
    double count;
    double publish_interval;
    ros::Subscriber num_sub;
    ros::Publisher count_pub;
    ros::Timer timer_pub;
    ros::ServiceServer reset_srv;
};

int main(int argc, char **argv) {
    ros::init(argc, argv, "counter_node");
    ros::NodeHandle nh;//deixando claro que questamos trabalhando com o namespace privado do ROS
    Counter counter(&nh);
    ros::spin();
    return 0;
}
```

counter_cpp.launch

```
<launch>
    <arg name = "number_topic" default= "/number"/>
    <param  name="initial_count" type = "double" value= "3.1415" />
    <param  name="pub_rate" type = "double" value= "2.5" />
    <param  name="delete_param" type = "bool" value= "True" />
    <node name = "counter_cpp" pkg = "my_project_cpp" type="counter" output="screen" >
        <remap from= "/number" to="$(arg number_topic)" />
    </node>
</launch>
```

17.3 adicionando parametros a um arquivo separado

counter_cpp.launch

```
<launch>
    <arg name = "number_topic" default= "/number"/>
    <!--param  name="initial_count" type = "double" value= "3.1415" />
    <param  name="pub_rate" type = "double" value= "2.5" /-->
    <param  name="delete_param" type = "bool" value= "True" />

    <rospram command="load" file = "$(my_project_bringup)/config/my_project.yaml"/>

    <node name = "counter_cpp" pkg = "my_project_cpp" type="counter" output="screen" >
        <remap from= "/number" to="$(arg number_topic)" />
    </node>
</launch>
```

my_project.yaml

```
initial_count: 5
pub_rate: 0.1
```

18 ROS parameter com python

prestar sempre atenção no namespace dos parametros.

counter.py

```
#!/usr/bin/env python3
```

```python
import rospy
from std_msgs.msg import Float64
from std_srvs.srv import Empty, EmptyResponse

class Counter:

    def __init__(self):
        self.count = rospy.get_param("initial_count", 0)
        self.publish_interval = rospy.get_param("~pub_rate", 3)


        if(rospy.has_param("~pub_rate") or rospy.has_param("initial_count")):
            rospy.loginfo("parametros customizados")
            rospy.set_param("~custom_param", True)
        else:
            rospy.set_param("~custom_param", False)

        if(rospy.has_param("~delete_param")):
            rospy.loginfo("deletando parametros")
            if(rospy.has_param("~pub_rate")):
                rospy.delete_param("~pub_rate")
            if(rospy.has_param("initial_count")):
                rospy.delete_param("initial_count")


        self.num_sub = rospy.Subscriber("/number", Float64, self.numberCallback, queue_size = 10)
        self.count_pub = rospy.Publisher("/current_count", Float64, queue_size = 10)
        self.timer_pub = rospy.Timer(rospy.Duration(self.publish_interval), self.timerCallback)
        self.reset_srv = rospy.Service('reset_counter', Empty, self.resetSrvCallback)

    def numberCallback(self, msg):
        self.count = self.count + msg.data
        rospy.loginfo("Contagem Atual " + str(self.count))


    def timerCallback(self, event):
        msg = Float64()
        msg.data = self.count
        self.count_pub.publish(msg)
    def resetSrvCallback(self, req):
        self.count = 0
        rospy.loginfo("resetando a contagem")
        return EmptyResponse()

if __name__ == '__main__':
    try:
        rospy.init_node("counter_node")
        Counter()
        rospy.spin()
    except rospy.ROSInterruptException:
        pass
```

counter_py.launch

```xml
    <launch>
        <arg name ="number_topic" default="/number"/>
        <param name="initial_count" type = "double" value ="-10.0"/>
        <node name = "counter_py" pkg ="my_project_py" type = "counter.py" output= "screen" >
            <param name="pub_rate" type = "double" value ="0.1"/>
             <param name="~delete_param" type = "bool" value ="false"/>
            <remap from="/number" to ="$(arg number_topic)"/>
        </node>

          <node name = "counter_py2" pkg ="my_project_py" type = "counter.py" output= "screen" >
            <param name="pub_rate" type = "double" value ="0.1"/>
            <remap from="/number" to ="$(arg number_topic)"/>
        </node>

        <!--node name = "num_pub    " pkg ="my_project_py" type = "counter.py" output= "screen" /
            -->


    </launch>
```

number_publisher.py

```python
#!/usr/bin/env python3
import rospy
from std_msgs.msg import Float64
from std_srvs.srv import Empty

class NumberPublisher:

    def __init__(self):
        self.number = rospy.get_param("num", 3.1)
        self.publish_interval = rospy.get_param("pub_interval", 1.5)
        self.reset_interval = rospy.get_param("reset", 8)

        self.num_pub = rospy.Publisher("/number", Float64, queue_size = 10)
```

```
        self.timer_pub = rospy.Timer(rospy.Duration(self.publish_interval), self.timerCallback)
        self.timer_reset = rospy.Timer(rospy.Duration(self.reset_interval), self.
            timerResetCallback)
        self.client_reset = rospy.ServiceProxy("reset_counter", Empty)

    def timerCallback(self, event):
        msg = Float64()
        msg.data = self.number
        self.num_pub.publish(msg)

    def timerResetCallback(self, event):
        rospy.loginfo("solicitacao de reset")
        self.client_reset()



if __name__ == '__main__':
    try:
        rospy.init_node("number_publisher_node")
        NumberPublisher()
        rospy.spin()
    except rospy.ROSInterruptException:
        pass
```

my_project_py.launch

```
<launch>

    <include file = "$(find my_project_bringup)/launch/counter_py.launch">
        <arg name ="number_topic" value = "new_number"/>

    </include>

    <node name= "num_pub_py" pkg ="my_project_py" type="number_publisher.py" output ="screen"/>
    <param name= "num"  type = "double" value = "4.5"/>
    <param name= "pub_interval" type = "double" value= "2.5"/>
    <param name= "reset" type= "double" value = "2"/>

</launch>
```

19 Custom Message

comando do terminal para o pacote de std msg

```
catkin_create_pkg my_project_msg std_msgs message_generation
```

configuração do cmake list interno do pkg de custom message

my_project_msg/CMakeLists.txt

```
cmake_minimum_required(VERSION 3.0.2)
project(my_project_msg)

## Compile as C++11, supported in ROS Kinetic and newer
# add_compile_options(-std=c++11)

## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
  message_generation
  std_msgs
)

add_message_files(
  FILES
  CounterHistory.msg

)
generate_messages(
  DEPENDENCIES
  std_msgs

)

catkin_package(
  CATKIN_DEPENDS message_generation std_msgs

)
```

foi adicionado <exec_depend>message_generation</exec_depend> pois o ROS tem dois packages com o mesmo nome

src/my_project_msg/package.xml

```
<?xml version="1.0"?>
<package format="2">
  <name>my_project_msg</name>
  <version>0.0.0</version>
```

```
  <description>The my_project_msg package</description>

  <!-- One maintainer tag required, multiple allowed, one person per tag -->
  <!-- Example:  -->
  <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
  <maintainer email="marcos@todo.todo">marcos</maintainer>


  <!-- One license tag required, multiple allowed, one license per tag -->
  <!-- Commonly used license strings: -->
  <!--   BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
  <license>TODO</license>


  <!-- Url tags are optional, but multiple are allowed, one per tag -->
  <!-- Optional attribute type can be: website, bugtracker, or repository -->
  <!-- Example: -->
  <!-- <url type="website">http://wiki.ros.org/my_project_msg</url> -->


  <!-- Author tags are optional, multiple are allowed, one per tag -->
  <!-- Authors do not have to be maintainers, but could be -->
  <!-- Example: -->
  <!-- <author email="jane.doe@example.com">Jane Doe</author> -->


  <!-- The *depend tags are used to specify dependencies -->
  <!-- Dependencies can be catkin packages or system dependencies -->
  <!-- Examples: -->
  <!-- Use depend as a shortcut for packages that are both build and exec dependencies -->
  <!--   <depend>roscpp</depend> -->
  <!--   Note that this is equivalent to the following: -->
  <!--   <build_depend>roscpp</build_depend> -->
  <!--   <exec_depend>roscpp</exec_depend> -->
  <!-- Use build_depend for packages you need at compile time: -->
  <!--   <build_depend>message_generation</build_depend> -->
  <!-- Use build_export_depend for packages you need in order to build against this package: -->
  <!--   <build_export_depend>message_generation</build_export_depend> -->
  <!-- Use buildtool_depend for build tool packages: -->
  <!--   <buildtool_depend>catkin</buildtool_depend> -->
  <!-- Use exec_depend for packages you need at runtime: -->
  <!--   <exec_depend>message_runtime</exec_depend> -->
  <!-- Use test_depend for packages you need only for testing: -->
  <!--   <test_depend>gtest</test_depend> -->
  <!-- Use doc_depend for packages you need only for building documentation: -->
  <!--   <doc_depend>doxygen</doc_depend> -->
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>message_generation</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_export_depend>std_msgs</build_export_depend>
  <exec_depend>std_msgs</exec_depend>
  <exec_depend>message_generation</exec_depend>


  <!-- The export tag contains other, unspecified, tags -->
  <export>
    <!-- Other tools can request additional information be placed here -->

  </export>
</package>
```

src/my_project_msg/msg/CounterHistory.msg

```
float64 current_value
float64 last_value
```

20 Custom message com c++

src/my_project_cpp/CMakeLists.txt

```
cmake_minimum_required(VERSION 3.0.2)
project(my_project_cpp)

## Compile as C++11, supported in ROS Kinetic and newer
# add_compile_options(-std=c++11)

## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
  roscpp
  std_msgs
  my_project_msg
)

## System dependencies are found with CMake's conventions
# find_package(Boost REQUIRED COMPONENTS system)
```

```
18
19
20  ## Uncomment this if the package has a setup.py. This macro ensures
21  ## modules and global scripts declared therein get installed
22  ## See http://ros.org/doc/api/catkin/html/user_guide/setup_dot_py.html
23  # catkin_python_setup()
24
25  ################################################
26  ## Declare ROS messages, services and actions ##
27  ################################################
28
29  ## To declare and build messages, services or actions from within this
30  ## package, follow these steps:
31  ## * Let MSG_DEP_SET be the set of packages whose message types you use in
32  ##   your messages/services/actions (e.g. std_msgs, actionlib_msgs, ...).
33  ## * In the file package.xml:
34  ##   * add a build_depend tag for "message_generation"
35  ##   * add a build_depend and a exec_depend tag for each package in MSG_DEP_SET
36  ##   * If MSG_DEP_SET isn't empty the following dependency has been pulled in
37  ##     but can be declared for certainty nonetheless:
38  ##     * add a exec_depend tag for "message_runtime"
39  ## * In this file (CMakeLists.txt):
40  ##   * add "message_generation" and every package in MSG_DEP_SET to
41  ##     find_package(catkin REQUIRED COMPONENTS ...)
42  ##   * add "message_runtime" and every package in MSG_DEP_SET to
43  ##     catkin_package(CATKIN_DEPENDS ...)
44  ##   * uncomment the add_*_files sections below as needed
45  ##     and list every .msg/.srv/.action file to be processed
46  ##   * uncomment the generate_messages entry below
47  ##   * add every package in MSG_DEP_SET to generate_messages(DEPENDENCIES ...)
48
49  ## Generate messages in the 'msg' folder
50  # add_message_files(
51  #   FILES
52  #   Message1.msg
53  #   Message2.msg
54  # )
55
56  ## Generate services in the 'srv' folder
57  # add_service_files(
58  #   FILES
59  #   Service1.srv
60  #   Service2.srv
61  # )
62
63  ## Generate actions in the 'action' folder
64  # add_action_files(
65  #   FILES
66  #   Action1.action
67  #   Action2.action
68  # )
69
70  ## Generate added messages and services with any dependencies listed here
71  # generate_messages(
72  #   DEPENDENCIES
73  #   std_msgs
74  # )
75
76  ################################################
77  ## Declare ROS dynamic reconfigure parameters ##
78  ################################################
79
80  ## To declare and build dynamic reconfigure parameters within this
81  ## package, follow these steps:
82  ## * In the file package.xml:
83  ##   * add a build_depend and a exec_depend tag for "dynamic_reconfigure"
84  ## * In this file (CMakeLists.txt):
85  ##   * add "dynamic_reconfigure" to
86  ##     find_package(catkin REQUIRED COMPONENTS ...)
87  ##   * uncomment the "generate_dynamic_reconfigure_options" section below
88  ##     and list every .cfg file to be processed
89
90  ## Generate dynamic reconfigure parameters in the 'cfg' folder
91  # generate_dynamic_reconfigure_options(
92  #   cfg/DynReconf1.cfg
93  #   cfg/DynReconf2.cfg
94  # )
95
96  ##################################
97  ## catkin specific configuration ##
98  ##################################
99  ## The catkin_package macro generates cmake config files for your package
100 ## Declare things to be passed to dependent projects
101 ## INCLUDE_DIRS: uncomment this if your package contains header files
102 ## LIBRARIES: libraries you create in this project that dependent projects also need
103 ## CATKIN_DEPENDS: catkin_packages dependent projects also need
104 ## DEPENDS: system dependencies of this project that dependent projects also need
105 catkin_package(
106 #  INCLUDE_DIRS include
```

```
#  LIBRARIES my_project_cpp
#  CATKIN_DEPENDS roscpp std_msgs
#  DEPENDS system_lib
)

############
## Build ##
############

## Specify additional locations of header files
## Your package locations should be listed before other locations
include_directories(
# include
  ${catkin_INCLUDE_DIRS}
)

## Declare a C++ library
# add_library(${PROJECT_NAME}
#   src/${PROJECT_NAME}/my_project_cpp.cpp
# )

## Add cmake target dependencies of the library
## as an example, code may need to be generated before libraries
## either from message generation or dynamic reconfigure
# add_dependencies(${PROJECT_NAME} ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})

## Declare a C++ executable
## With catkin_make all packages are built within a single CMake context
## The recommended prefix ensures that target names across packages don't collide
# add_executable(${PROJECT_NAME}_node src/my_project_cpp_node.cpp)

## Rename C++ executable without prefix
## The above recommended prefix causes long target names, the following renames the
## target back to the shorter version for ease of user use
## e.g. "rosrun someones_pkg node" instead of "rosrun someones_pkg someones_pkg_node"
# set_target_properties(${PROJECT_NAME}_node PROPERTIES OUTPUT_NAME node PREFIX "")

## Add cmake target dependencies of the executable
## same as for the library above
# add_dependencies(${PROJECT_NAME}_node ${${PROJECT_NAME}_EXPORTED_TARGETS} ${
    catkin_EXPORTED_TARGETS})

## Specify libraries to link a library or executable target against
# target_link_libraries(${PROJECT_NAME}_node
#   ${catkin_LIBRARIES}
# )

#############
## Install ##
#############

# all install targets should use catkin DESTINATION variables
# See http://ros.org/doc/api/catkin/html/adv_user_guide/variables.html

## Mark executable scripts (Python etc.) for installation
## in contrast to setup.py, you can choose the destination
# catkin_install_python(PROGRAMS
#   scripts/my_python_script
#   DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
# )

## Mark executables for installation
## See http://docs.ros.org/melodic/api/catkin/html/howto/format1/building_executables.html
# install(TARGETS ${PROJECT_NAME}_node
#   RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
# )

## Mark libraries for installation
## See http://docs.ros.org/melodic/api/catkin/html/howto/format1/building_libraries.html
# install(TARGETS ${PROJECT_NAME}
#   ARCHIVE DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
#   LIBRARY DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
#   RUNTIME DESTINATION ${CATKIN_GLOBAL_BIN_DESTINATION}
# )

## Mark cpp header files for installation
# install(DIRECTORY include/${PROJECT_NAME}/
#   DESTINATION ${CATKIN_PACKAGE_INCLUDE_DESTINATION}
#   FILES_MATCHING PATTERN "*.h"
#   PATTERN ".svn" EXCLUDE
# )

## Mark other files for installation (e.g. launch and bag files, etc.)
# install(FILES
#   # myfile1
#   # myfile2
#   DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}
# )

```

```
##############
## Testing ##
##############

## Add gtest based cpp test target and link libraries
# catkin_add_gtest(${PROJECT_NAME}-test test/test_my_project_cpp.cpp)
# if(TARGET ${PROJECT_NAME}-test)
#   target_link_libraries(${PROJECT_NAME}-test ${PROJECT_NAME})
# endif()

## Add folders to be run by python nosetests
# catkin_add_nosetests(test)
add_executable(counter src/counter.cpp)
target_link_libraries(counter ${catkin_LIBRARIES})

add_executable(number_publisher src/number_publisher.cpp)
target_link_libraries(number_publisher ${catkin_LIBRARIES})
```

src/my_project_cpp/package.xml

```
<?xml version="1.0"?>
<package format="2">
  <name>my_project_cpp</name>
  <version>0.0.0</version>
  <description>The my_project_cpp package</description>

  <!-- One maintainer tag required, multiple allowed, one person per tag -->
  <!-- Example:  -->
  <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
  <maintainer email="marcos@todo.todo">marcos</maintainer>


  <!-- One license tag required, multiple allowed, one license per tag -->
  <!-- Commonly used license strings: -->
  <!--   BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
  <license>TODO</license>


  <!-- Url tags are optional, but multiple are allowed, one per tag -->
  <!-- Optional attribute type can be: website, bugtracker, or repository -->
  <!-- Example: -->
  <!-- <url type="website">http://wiki.ros.org/my_project_cpp</url> -->


  <!-- Author tags are optional, multiple are allowed, one per tag -->
  <!-- Authors do not have to be maintainers, but could be -->
  <!-- Example: -->
  <!-- <author email="jane.doe@example.com">Jane Doe</author> -->


  <!-- The *depend tags are used to specify dependencies -->
  <!-- Dependencies can be catkin packages or system dependencies -->
  <!-- Examples: -->
  <!-- Use depend as a shortcut for packages that are both build and exec dependencies -->
  <!--   <depend>roscpp</depend> -->
  <!--   Note that this is equivalent to the following: -->
  <!--   <build_depend>roscpp</build_depend> -->
  <!--   <exec_depend>roscpp</exec_depend> -->
  <!-- Use build_depend for packages you need at compile time: -->
  <!--   <build_depend>message_generation</build_depend> -->
  <!-- Use build_export_depend for packages you need in order to build against this package: -->
  <!--   <build_export_depend>message_generation</build_export_depend> -->
  <!-- Use buildtool_depend for build tool packages: -->
  <!--   <buildtool_depend>catkin</buildtool_depend> -->
  <!-- Use exec_depend for packages you need at runtime: -->
  <!--   <exec_depend>message_runtime</exec_depend> -->
  <!-- Use test_depend for packages you need only for testing: -->
  <!--   <test_depend>gtest</test_depend> -->
  <!-- Use doc_depend for packages you need only for building documentation: -->
  <!--   <doc_depend>doxygen</doc_depend> -->
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_depend>my_project_msg</build_depend>
  <build_export_depend>roscpp</build_export_depend>
  <build_export_depend>std_msgs</build_export_depend>
  <build_export_depend>my_project_msg</build_export_depend>
  <exec_depend>roscpp</exec_depend>
  <exec_depend>std_msgs</exec_depend>
  <exec_depend>my_project_msg</exec_depend>


  <!-- The export tag contains other, unspecified, tags -->
  <export>
    <!-- Other tools can request additional information be placed here -->

  </export>
</package>
```

No momento do curso ele nao entrava na funcao de callback portanto nao atualizava os parametros da funcao, no terminal.

src/my_project_cpp/src/counter.cpp

```cpp
#include "ros/ros.h"

// Messagens
#include "std_msgs/Float64.h"
#include "std_srvs/Empty.h"
#include "my_project_msg/CounterHistory.h"


class Counter{

    public:
        Counter(ros::NodeHandle *nh){
            count = 0;
            publish_interval = 1;

            if (nh->getParam("initial_count", count) ){
                ROS_INFO("Contagem inicial em %f", count);
            }
            else{
                ROS_INFO("Contagem inicial nao definido");
            }
            nh->param<double>("pub_rate", publish_interval, 1);

            if (nh->hasParam("pub_rate") || nh->hasParam("initial_count")  ){
                ROS_WARN("Parametros customizados");
                nh->setParam("custom_param", true);
            }
            else{
                nh->setParam("custom_param", false);
            }

            if (nh->hasParam("delete_param")){
                ROS_ERROR("Deletando Parametros");
                nh->deleteParam("pub_rate");
                nh->deleteParam("initial_count");
            }

            last_count = count;
            cycles = 0;

            num_sub = nh->subscribe("/number", 10, &Counter::numberCallback, this);
            count_pub = nh->advertise<std_msgs::Float64>("/current_count", 10);

            timer_pub = nh->createTimer(ros::Duration(publish_interval), &Counter::timerCallback, this);
            reset_srv = nh->advertiseService("/reset_counter", &Counter::resetSrvCallback, this);

            history_pub = nh->advertise<my_project_msg::CounterHistory>("/history_counter", 10);
        }

        void numberCallback(const std_msgs::Float64 &msg){
            ROS_INFO("Callback chamado com valor: %f", msg.data);
            cycles++;
            last_count = count;
            count += msg.data;
            ROS_INFO("Contagem Atual %f", count);
        }

        void timerCallback(const ros::TimerEvent &event){
            ROS_INFO("Timer chamado");
            std_msgs::Float64 msg;
            msg.data = count;
            count_pub.publish(msg);

            my_project_msg::CounterHistory history_msg;
            history_msg.current_value = count;
            history_msg.last_value = last_count;
            history_msg.cycles = cycles;
            history_pub.publish(history_msg);
        }

        bool resetSrvCallback(std_srvs::Empty::Request &req, std_srvs::Empty::Response &res){
            count = 0;
            ROS_INFO("Resetando a contagem.");
            return true;
        }


    private:
        double count;
        double publish_interval;
        double last_count;
        int cycles;

        ros::Publisher history_pub;
        ros::Subscriber num_sub;
```

```
        ros::Publisher count_pub;
        ros::Timer timer_pub;
        ros::ServiceServer reset_srv;
        ros::ServiceServer check_greater_srv;
};


int main(int argc, char **argv){

    ros::init(argc, argv, "counter_node");
    ros::NodeHandle nh;
    Counter counter = Counter(&nh);
    ros::spin();

    return 0;
}
```

21 Custom message em python

O codigo nao entra no callback, portanto não atualiza os parametros

src/my_project_py/scripts/counter.py

```
#!/usr/bin/env python3
import rospy
from std_msgs.msg import Float64
from std_srvs.srv import Empty, EmptyResponse
from my_project_msg.msg import CounterHistory

class Counter:

    def __init__(self):
        self.count = rospy.get_param("initial_count", 0)
        self.publish_interval = rospy.get_param("~pub_rate", 3)


        if(rospy.has_param("~pub_rate") or rospy.has_param("initial_count")):
            rospy.loginfo("parametros customizados")
            rospy.set_param("~custom_param", True)
        else:
            rospy.set_param("~custom_param", False)

        if(rospy.has_param("~delete_param")):
            rospy.loginfo("deletando parametros")
            if(rospy.has_param("~pub_rate")):
                rospy.delete_param("~pub_rate")
            if(rospy.has_param("initial_count")):
                rospy.delete_param("initial_count")
        self.last_count = self.count
        self.cycles = 0

        self.num_sub = rospy.Subscriber("/number", Float64, self.numberCallback, queue_size = 10)
        self.count_pub = rospy.Publisher("/current_count", Float64, queue_size = 10)
        self.timer_pub = rospy.Timer(rospy.Duration(self.publish_interval), self.timerCallback)
        self.reset_srv = rospy.Service('reset_counter', Empty, self.resetSrvCallback)
        self.history_pub = rospy.Publisher("/history_count", CounterHistory, queue_size = 10)

    def numberCallback(self, msg):
        self.cycles = self.cycles + 1
        self.last_count = self.count
        self.count = self.count + msg.data
        rospy.loginfo("Contagem Atual " + str(self.count))


    def timerCallback(self, event):
        msg = Float64()
        msg.data = self.count
        self.count_pub.publish(msg)
        history_msg = CounterHistory()
        history_msg.current_value = self.count
        history_msg.last_value = self.last_count
        history_msg.cycles = self.cycles
        self.history_pub.publish(history_msg)

    def resetSrvCallback(self, req):
        self.count = 0
        rospy.loginfo("resetando a contagem")
        return EmptyResponse()

if __name__ == '__main__':
    try:
        rospy.init_node("counter_node")
        Counter()
        rospy.spin()
    except rospy.ROSInterruptException:
        pass
```

src/my_project_py/package.xml

```
<?xml version="1.0"?>
<package format="2">
  <name>my_project_py</name>
  <version>0.0.0</version>
  <description>The my_project_py package</description>

  <!-- One maintainer tag required, multiple allowed, one person per tag -->
  <!-- Example:  -->
  <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
  <maintainer email="marcos@todo.todo">marcos</maintainer>


  <!-- One license tag required, multiple allowed, one license per tag -->
  <!-- Commonly used license strings: -->
  <!--   BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
  <license>TODO</license>


  <!-- Url tags are optional, but multiple are allowed, one per tag -->
  <!-- Optional attribute type can be: website, bugtracker, or repository -->
  <!-- Example: -->
  <!-- <url type="website">http://wiki.ros.org/my_project_py</url> -->


  <!-- Author tags are optional, multiple are allowed, one per tag -->
  <!-- Authors do not have to be maintainers, but could be -->
  <!-- Example: -->
  <!-- <author email="jane.doe@example.com">Jane Doe</author> -->


  <!-- The *depend tags are used to specify dependencies -->
  <!-- Dependencies can be catkin packages or system dependencies -->
  <!-- Examples: -->
  <!-- Use depend as a shortcut for packages that are both build and exec dependencies -->
  <!--   <depend>roscpp</depend> -->
  <!--   Note that this is equivalent to the following: -->
  <!--   <build_depend>roscpp</build_depend> -->
  <!--   <exec_depend>roscpp</exec_depend> -->
  <!-- Use build_depend for packages you need at compile time: -->
  <!--   <build_depend>message_generation</build_depend> -->
  <!-- Use build_export_depend for packages you need in order to build against this package: -->
  <!--   <build_export_depend>message_generation</build_export_depend> -->
  <!-- Use buildtool_depend for build tool packages: -->
  <!--   <buildtool_depend>catkin</buildtool_depend> -->
  <!-- Use exec_depend for packages you need at runtime: -->
  <!--   <exec_depend>message_runtime</exec_depend> -->
  <!-- Use test_depend for packages you need only for testing: -->
  <!--   <test_depend>gtest</test_depend> -->
  <!-- Use doc_depend for packages you need only for building documentation: -->
  <!--   <doc_depend>doxygen</doc_depend> -->
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_depend>my_project_msg</build_depend>
  <build_export_depend>rospy</build_export_depend>
  <build_export_depend>std_msgs</build_export_depend>
  <build_export_depend>my_project_msg</build_export_depend>
  <exec_depend>rospy</exec_depend>
  <exec_depend>std_msgs</exec_depend>
  <exec_depend>my_project_msg</exec_depend>



  <!-- The export tag contains other, unspecified, tags -->
  <export>
    <!-- Other tools can request additional information be placed here -->

  </export>
</package>
```

src/my_project_py/CMakeLists.txt

```
cmake_minimum_required(VERSION 3.0.2)
project(my_project_py)

## Compile as C++11, supported in ROS Kinetic and newer
# add_compile_options(-std=c++11)

## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
  rospy
  std_msgs
  my_project_msg
)

## System dependencies are found with CMake's conventions
```

```
# find_package(Boost REQUIRED COMPONENTS system)


## Uncomment this if the package has a setup.py. This macro ensures
## modules and global scripts declared therein get installed
## See http://ros.org/doc/api/catkin/html/user_guide/setup_dot_py.html
# catkin_python_setup()

################################################
## Declare ROS messages, services and actions ##
################################################

## To declare and build messages, services or actions from within this
## package, follow these steps:
## * Let MSG_DEP_SET be the set of packages whose message types you use in
##   your messages/services/actions (e.g. std_msgs, actionlib_msgs, ...).
## * In the file package.xml:
##   * add a build_depend tag for "message_generation"
##   * add a build_depend and a exec_depend tag for each package in MSG_DEP_SET
##   * If MSG_DEP_SET isn't empty the following dependency has been pulled in
##     but can be declared for certainty nonetheless:
##     * add a exec_depend tag for "message_runtime"
## * In this file (CMakeLists.txt):
##   * add "message_generation" and every package in MSG_DEP_SET to
##     find_package(catkin REQUIRED COMPONENTS ...)
##   * add "message_runtime" and every package in MSG_DEP_SET to
##     catkin_package(CATKIN_DEPENDS ...)
##   * uncomment the add_*_files sections below as needed
##     and list every .msg/.srv/.action file to be processed
##   * uncomment the generate_messages entry below
##   * add every package in MSG_DEP_SET to generate_messages(DEPENDENCIES ...)

## Generate messages in the 'msg' folder
# add_message_files(
#   FILES
#   Message1.msg
#   Message2.msg
# )

## Generate services in the 'srv' folder
# add_service_files(
#   FILES
#   Service1.srv
#   Service2.srv
# )

## Generate actions in the 'action' folder
# add_action_files(
#   FILES
#   Action1.action
#   Action2.action
# )

## Generate added messages and services with any dependencies listed here
# generate_messages(
#   DEPENDENCIES
#   std_msgs
# )

################################################
## Declare ROS dynamic reconfigure parameters ##
################################################

## To declare and build dynamic reconfigure parameters within this
## package, follow these steps:
## * In the file package.xml:
##   * add a build_depend and a exec_depend tag for "dynamic_reconfigure"
## * In this file (CMakeLists.txt):
##   * add "dynamic_reconfigure" to
##     find_package(catkin REQUIRED COMPONENTS ...)
##   * uncomment the "generate_dynamic_reconfigure_options" section below
##     and list every .cfg file to be processed

## Generate dynamic reconfigure parameters in the 'cfg' folder
# generate_dynamic_reconfigure_options(
#   cfg/DynReconf1.cfg
#   cfg/DynReconf2.cfg
# )

###################################
## catkin specific configuration ##
###################################
## The catkin_package macro generates cmake config files for your package
## Declare things to be passed to dependent projects
## INCLUDE_DIRS: uncomment this if your package contains header files
## LIBRARIES: libraries you create in this project that dependent projects also need
## CATKIN_DEPENDS: catkin_packages dependent projects also need
## DEPENDS: system dependencies of this project that dependent projects also need
catkin_package(
```

```
#  INCLUDE_DIRS include
#  LIBRARIES my_project_py
#  CATKIN_DEPENDS rospy std_msgs
#  DEPENDS system_lib
)

############
## Build ##
############

## Specify additional locations of header files
## Your package locations should be listed before other locations
include_directories(
# include
  ${catkin_INCLUDE_DIRS}
)

## Declare a C++ library
# add_library(${PROJECT_NAME}
#   src/${PROJECT_NAME}/my_project_py.cpp
# )

## Add cmake target dependencies of the library
## as an example, code may need to be generated before libraries
## either from message generation or dynamic reconfigure
# add_dependencies(${PROJECT_NAME} ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})

## Declare a C++ executable
## With catkin_make all packages are built within a single CMake context
## The recommended prefix ensures that target names across packages don't collide
# add_executable(${PROJECT_NAME}_node src/my_project_py_node.cpp)

## Rename C++ executable without prefix
## The above recommended prefix causes long target names, the following renames the
## target back to the shorter version for ease of user use
## e.g. "rosrun someones_pkg node" instead of "rosrun someones_pkg someones_pkg_node"
# set_target_properties(${PROJECT_NAME}_node PROPERTIES OUTPUT_NAME node PREFIX "")

## Add cmake target dependencies of the executable
## same as for the library above
# add_dependencies(${PROJECT_NAME}_node ${${PROJECT_NAME}_EXPORTED_TARGETS} ${
    catkin_EXPORTED_TARGETS})

## Specify libraries to link a library or executable target against
# target_link_libraries(${PROJECT_NAME}_node
#   ${catkin_LIBRARIES}
# )

#############
## Install ##
#############

# all install targets should use catkin DESTINATION variables
# See http://ros.org/doc/api/catkin/html/adv_user_guide/variables.html

## Mark executable scripts (Python etc.) for installation
## in contrast to setup.py, you can choose the destination
# catkin_install_python(PROGRAMS
#   scripts/my_python_script
#   DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
# )

## Mark executables for installation
## See http://docs.ros.org/melodic/api/catkin/html/howto/format1/building_executables.html
# install(TARGETS ${PROJECT_NAME}_node
#   RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
# )

## Mark libraries for installation
## See http://docs.ros.org/melodic/api/catkin/html/howto/format1/building_libraries.html
# install(TARGETS ${PROJECT_NAME}
#   ARCHIVE DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
#   LIBRARY DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
#   RUNTIME DESTINATION ${CATKIN_GLOBAL_BIN_DESTINATION}
# )

## Mark cpp header files for installation
# install(DIRECTORY include/${PROJECT_NAME}/
#   DESTINATION ${CATKIN_PACKAGE_INCLUDE_DESTINATION}
#   FILES_MATCHING PATTERN "*.h"
#   PATTERN ".svn" EXCLUDE
# )

## Mark other files for installation (e.g. launch and bag files, etc.)
# install(FILES
#   # myfile1
#   # myfile2
#   DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}
# )
```

```
194
195 ##############
196 ## Testing ##
197 ##############
198
199 ## Add gtest based cpp test target and link libraries
200 # catkin_add_gtest(${PROJECT_NAME}-test test/test_my_project_py.cpp)
201 # if(TARGET ${PROJECT_NAME}-test)
202 #    target_link_libraries(${PROJECT_NAME}-test ${PROJECT_NAME})
203 # endif()
204
205 ## Add folders to be run by python nosetests
206 # catkin_add_nosetests(test)
```

22 Custom Service

O diretorio para o service é criado dentro do pkg de msg pelo serviço se assemelhar. Adiciona-se o service no cmake

my_project$_{m}sg/srv/CheckNumber.srv$

```
1 float64 number
2 ---
3 bool result
```

my$_{p}roject_{m}sg/CMakeLists.txt)project(my_project_msg)$

```
1
2 ## Compile as C++11, supported in ROS Kinetic and newer
3 # add_compile_options(-std=c++11)
4
5 \#\# Find catkin macros and libraries
6 ## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
7 ## is used, also find other catkin packages
8 find_package(catkin REQUIRED COMPONENTS
9   message_generation
10  std_msgs
11
12 )
13 add_message_files(
14   FILES
15   CounterHistory.msg
16
17 )
18 add_service_files(
19   FILES
20   CheckNumber.srv
21
22 )
23
24 generate_messages(
25   DEPENDENCIES
26   std_msgs
27
28 )
29
30 catkin_package(
31   CATKIN_DEPENDS message_generation std_msgs
32
33 )
```

terminal bash

```
1 rossrv package my_project_msg
2 rossrv show my_project_msg/CheckNumber.srv
```

23

src/my_project_py/CMakeLists.txth

```
```