Documentação de Software Desenvolvido

LT1000: tratamento das medições

Descrição de Tarefa

Informação Genérica

Nome da tarefa	LT1000
Descrição	Recebe as mensagens NMEA de um módulo de Hardware ou Simulador, consoante a especificação do Device. Descodifica a informação e envia mensagens IMC do tipo GPSFix and EulerAngle
Localização	source/src/Sensors/LT1000
Mensagens Recebidas	NMEA
Mensagens Enviadas	IMC::GPSFix IMC::EulerAngle
Parâmetros	Serial Port - Device = tcp://127.0.0.1:3000 Ou Serial Port - Device = /dev/ttyS0 Serial Port - Baud Rate = 115200 Sentence Order = GPZDA, GPGGA, GPRMC, GPVTG, GPHDT

Processo de Setup

Inicialização da Tarefa

Faz a inicialização de todas as variáveis tendo em conta os parâmetros passados.
Inicializam-se os consumidores para as mensagens de input NMEA.

Main

A tarefa é reativa, pelo que o main apenas recebe as mensagens IMC externas.

Receção de Mensagens NMEA

A receção de mensagens faz-se na função onResourceAcquisition(void), quer esteja ligado no simulador quer esteja ligado ao módulo de sensores LT1000. Consoante o código do tipo de mensagem, é invocada uma função própria para descodificação dos vários campos associados à norma.

Teste Unitário

Para testar o módulo de sensorização, utilizou-se primeiramente um simulador de mensagens NMEA e depois um GPS e um LT1000.

Resultado

Verifica-se que as mensagens são corretamente recebidas, descodificadas e transformadas em mensagens IMC próprias para uso do sistema.
Verifica-se um erro máximo de 6m face à localização atual devido a erros de conversão entre sistemas de coordenadas.

Comunicação UART Dune-Arduino

Descrição de Tarefa

Informação Genérica

Nome da tarefa	UART_Comm
Descrição	Recebe a posição dos lemes e o valor de atuação de cada motor, formata os dados para serem compreendidos pelo protocolo de comunicação no lado do Arduino e envia esses dados por UART via USB. As mensagens enviadas ao Arduino estão no formato (“Id do atuador no arduino” + “valor a atualizar no atuador”) e o data type em que as mensagens são enviadas é um array de **const char.**
Localização	src/Actuators/
Mensagens Recebidas	IMC::SetThrusterActuation IMC::SetServoPosition
Mensagens Enviadas	As mensagens enviadas não se enquadram no protocolo IMC, correspondem à escrita na porta série, com o formato já citado.

Parâmetros	**Valor Default**	**Descrição**
Target Producer	“Producer”	Target producer to read from
Serial Port - Device	"/dev/ttyACM0"	Serial port device (used to

		communicate with the actuator)
Serial Port - Baud Rate	112500	Serial port baud rate
Message Frequency	1.0	Frequency at which we send Thruster and Rudder Control Messages to Arduino
Actuation Lower Bound	1000	Received actuation values will be trimmed to this bound
Actuation Upper Bound	2000	Received actuation values will be trimmed to this bound

Variáveis Auxiliares

De maneira a evitar a criação e envio de comandos redundantes, foram utilizadas 2 variavéis auxiliares por atuador, uma variável que armazena o valor que foi atribuido para o último comando criado para esse atuador (para referência: **m_<atuador>_prev**) e uma variável que armazena o valor contido no último comando enviado ao atuador (para referência: **m_<atuador>_sent_prev**).

Processo de Setup

Inicialização da Tarefa

Para além de se declararem e atribuírem os parâmetros externos às devidas variáveis, são também criados comandos para todos os motores com o valor correspondente, nos Thrusters, ao estado estático (parado) e nos servos à posição inicial (**valor = 1500**).

Esses valores são guardados nas respectivas variáveis auxiliares do tipo **m_<atuador>_sent_prev**. Por fim são subscritas as mensagens IMC de entrada

onResourceAcquisition

Cria um objeto da Classe SerialPort e habilita a comunicação na porta série e com o Baud Rate definidos na inicialização.

onResourceRelease

Limpa a variável onde foi instanciado o objeto da Classe SerialPort.

onUpdateParameters

Monitoriza se o parâmetro de “Message Frequency” muda. No caso de mudar, dá set ao novo valor de Frequência.

Receção de Mensagens IMC

IMC::SetThrusterActuation e IMC::SetServoPosition

A receção de mensagens IMC é realizada pela função **consume** correspondente a cada um dos 2 tipos de mensagens.

As mensagens recebidos pela task contêm valores entre -1 e 1. Ao serem recebidos os valores sofrem uma conversão para o intervalo de 1000 a 2000, que corresponde à gama de valores utilizada pelo arduino para controlar os motores.

De seguida a task verifica se os valores se encontram dentro dos limites estabelecidos. Se os valores se encontrarem forem dos limites eles tomam o valor do limite mais próximo.

Por fim compara com os valores do ciclo anterior e no caso de serem diferentes, atualiza a respetiva variável auxiliar do tipo **m_<atuador>_prev** e cria um comando com esses valores

para o devido atuador através da função **createCommand**. (No caso do atuador a que se destina a mensagem ser um servo, é criado um comando para cada servo).

Processo Contínuo

Task

Verifica se o ciclo atual de execução ultrapassou o periodo por ciclo mais a tolerância (tolerância definida internamente no programa), no caso afirmativo, pressupõe que o buffer está cheio e limpa o mesmo.

Verifica também se os comando criados para enviar a cada motor (correspondentes aos que estão armazenados nas variáveis **m_<atuador>_prev**) são diferentes dos que foram previamente enviados (correspondentes aos das variáveis **m_<atuador>_sent_prev)** e no caso de serem diferentes executa a função **sendCommand** para enviá-los e atualiza as variavéis **m_<atuador>_sent_prev**.

Identificadores dos atuadores no Arduino

ID	Atuador
m	Thruster 1
M	Thruster 2
l	Rudder 1
L	Rudder 2

Teste Unitário

Para testar esta componente isoladamente foi desenvolvida outra tarefa (“Actuators/Value_Generator”) que gera valores entre -1 e 1 (valores que são incrementados a uma taxa de 0.1 a cada ciclo) para a atuação dos motores e dos lemes e envia esses valores para a tarefa **UART_Comm**.

Esses valores sofrem a devida conversão e formatação para comandos que são posteriormentes enviados para o arduino que realiza a atuação dos lemes em 2 motores servos e a atuação dos thrusters é simulado por 2 leds RGB que mudam de cor consoante o valor de atuação.

Funções

void consume (const IMC::SetThrusterActuation* msg)

void consume (const IMC::SetServoPosition* msg)

msg - > Mensagem IMC do tipo especificado.

bool serialPortIsInvalid()

Verifica se a porta série já foi inicializada, tentando inicializá-la se já não o estiver. Retorna **false** se a porta **estiver inicializada**. Retorna **true** se a porta **não estiver inicializada** (mesmo após tentativa de inicializá-la).

void createCommand(const std::string& cmd_type, fp32_t val)

cmd_type -> Tipo de Comando (Identificador do atuador no Arduino);

val -> Valor a ser atualizar no atuador.

Descrição: Cria um comando com o formato (Tipo de Comando+Valor) e armazena-o numa variável criada para o armazenamento dos comandos de cada atuador em especifico até que seja realizado o envio do mesmo.

void sendCommand(const char* cmd)

cmd -> Mensagem a enviar.

Descrição: Escreve a mensagem do comando na porta série.

A função avalia, previamente ao envio da mensagem, se a porta série está devidamente configurada através da função **serialPortIsInvalid()**.

Após algumas mensagens enviadas, o buffer da porta série é limpo.

Comunicação LoRa

Informação Genérica

Nome da tarefa	Lora_serial
Descrição	Recebe as mensagens dos planos simplificados vindas por serial no formato json do módulo LoRa6500Pro para que depois se possa fazer parse. As mensagens do estado da embarcação são enviadas através de serial, por USB-TTL, para o módulo LoRa6500Pro para que sejam enviadas por protocolo LoRa para a GCS.
Localização	src/Transports/
Mensagens Recebidas	char* Plan
Mensagens Enviadas	char* State

Parâmetros	**Valor Default**	**Descrição**
Entity Label	“Producer2”	Atribuição de entidade como Produtor
Serial Port - Device	"/dev/ttyS0"	Serial port device (usado para comunicar com o LoRa6500Pro)
Serial Port - Baud Rate	9600	Serial port baud rate

Processo de Setup

Inicialização da Tarefa

Declara-se e atribui-se os parâmetros externos às devidas variáveis.

Funções

void onResourceAcquisition (void)

Cria um objeto da Classe SerialPort chamado “m_uart” e habilita a comunicação na porta série e com o Baud Rate definidos na inicialização.

void onResourceRelease (void)

Limpa a variável onde foi instanciado o objeto da Classe SerialPort.

Processo Contínuo

Funções

void onDataTransmission (const char* State, unsigned int n)

Descrição: Escreve a mensagem de estado simplificado na porta série.

Utilizando o objeto criado da Classe SerialPort “m_uart” envia a mensagem do estado pela porta série para o LoRa6500Pro.

void onDataReception (char* Plan, unsigned int n, double timeout)

Descrição: Recebe a mensagem do plano simplificado da porta série e realiza o parsing.

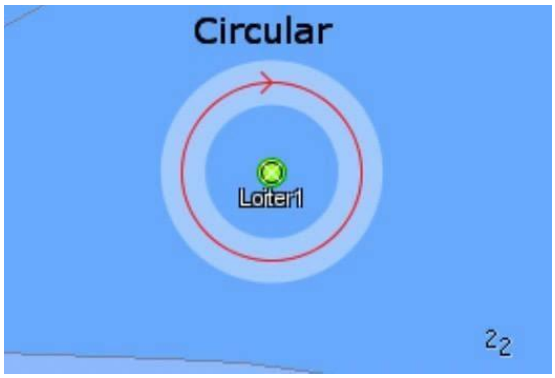
A função avalia, previamente, se na conexão realizada se alguma mensagem é recebida dentro de um certo timeout estabelecido, recorrendo ao valor retornado da função “Poll::poll(*m_uart, timeout)” .

De seguida, utilizando o objeto “m_uart” lê a mensage e atribui a string recebida para a variável “Plan” e retorna o número de caractéres recebidos para a variável “n_r”. Em caso de ocorrência de exceção liberta uma mensagem de erro de leitura.

Caso a leitura tenha sido bem sucedida verifica se os caracteres recebidos são superiores a 0. Caso não se verifica liberta uma mensagem de erro desconhecido e retorna.

No final o parsing é dado pela função “handleData(m_parser, Plan, n_r)”

Parametros do plano de manobra simplificado (Exemplo de uma Loiter maneuver)



Label	Descrição
ID	Identificação da manobra
Latitude	Latitude do ponto central da manobra
Longitude	Longitude do ponto central da manobra
Duration	Duração máxima da manobra em segundos
Loiter type	Tipo de manobra a executar (ex: Circular)

Speed	Velocidade na manobra em Nós
Length	Tipo de manobra a executar (ex: Circular)
Radius	Raio de distância desde o centro da manobra

Parametros do estado simplificado

Label	**Descrição**
ID	Identificação da embarcação
ID Manobra em curso	Identificação da manobra em execução
Location	Coordenadas geográficas da manobra em curso
Latitude	Latitude da embarcação
Longitude	Longitude da embarcação
Z	Profundidade da embarcação
Speed	Velocidade da embarcação em Nós

Yaw	Ângulo de rotação em relação ao eixo horizontal da embarcação
Pitch	Ângulo de rotação em relação ao eixo transversal da embarcação
Roll	Ângulo de rotação em relação ao eixo longitudinal da embarcação
Hora e data	Data e hórario do relógio da embarcação

Teste sem o módulo LoRa

Para testar esta tarefa foi desenvolvido uma interface de comunicação com o auxilio da aplicação de terminal Socat para simular o envio de mensagem por UART. Para tal é necessário proceder à instalação, caso já não o tenha, e executar o seguinte comando no terminal:

socat -d pty,raw,echo=0

Isto gerará uma porta USB virtual a qual se pode comunicar utilizando o seguinte comando:

echo "*aqui mensagem*" > /dev/pts/*aqui número porta gerada*

Caso a manta esteja a debitar valores de mensagens recebidas, a comunicação foi bem sucedida.

Configuração do módulo LoRa

Consultando a datasheet do aparelho é possível encontrar as nomenclaturas e funcionalidades de cada um dos pinos do terminal.

Definição	Definição
VCC	Fonte de Tensão Positiva
TXD	Conexão Tx para RS232 e TTL
RXD	Conexão Rx para RS232 e TTL
GND	Ground
A+	Conexão A+ para RS485
B-	Conexão B- para RS485
SET	Para dar Enable ao Modo de Configuração.\\ LOW para entrar no modo e HIGH para sair do modo
CS	Para dar Enable ao Modo de Operação.\\ LOW para entrar no modo sleep e HIGH para operação normal

Para que se possa fazer configuração do LoRa6500Pro, deve ser deixado no valor lógico LOW e assim entrar no modo configuração. Após estar neste modo é possível configurar o módulo ou com envio de comandos, tal como especificado na Datasheet, ou com o executável "LoRaPROV3.0.exe" fornecido pela GNiceRF, Figura seguinte.

Legenda: Executável "LoRaPROV3.0.exe" , fornecido pela GNiceRF

A Comunicação entre os módulos LoRa6500Pro foi estabelecida usando os protocolos de comunicação série TTL, por usb no caso da GCS, e RS232 no USV.

O baudrate escolhido para os módulos foi o recomendado de 9600bps, 8 bits de dados, 1 bit de paridade Par para deteção de erros e 2 stop bits.

Os protocolos de comunicação série são usados para enviar os planos para os módulos para eles depois enviarem via protocolo LoRa/LoRaWAN.

O modo de conexão entre os módulos está configurado como nó a nó, visto que a função trata-se por basicamente fazer uma ligação ponto a ponto entre a GCS e o USV.

A comunicação está segundo as normas europeias do European Telecommunications Standards Institute (ETSI), que estabelecem que para o LoRa os sinais RF devem ter 868MHz de frequência para evitar interferências devido ao seu grande alcance e com o mesmo NET ID estipulado para que não interferira com outros módulos LoRa. O canal e a banda devem ser os mesmos para que a comunicação seja feita com sucesso, os valores escolhidos foram 20 e 868 respetivamente. O Node ID deve ser distinto para que se faça a distinção entre os nós. Para o Spreading Factor (Power) utilizou-se o valor de 7, pois é suficiente para os testes realizados.

Teste com o módulo LoRa

Para a situação em que a manta utiliza o método Lora no envio das tarefas, a tarefa consegue eficazmente fazer o envio, receção e parsing da informação e constitui a prova de conceito para o envio do plano do Neptus (por mensagem IMC) para o Dune.

Controlo PID

Informação Genérica

Nome da tarefa	SimpleHeadingAndSpeed
Descrição	Recebe direção e velocidade (atuais e desejados) e realiza controlo PID de forma a obter um output de controlo que minimize o erro entre os dados recebidos. Esta tarefa é composta por dois controladores PID (um para a velocidade, outro para a direção) cujos outputs são combinados para se obter o controlo de motores. O controlo dos lemes apenas se aplica para o controlador PID de direção.d
Localização	src/Control/USV/
Mensagens Recebidas	IMC::EstimatedState IMC::DesiredHeading IMC::DesiredSpeed IMC::Abort IMC::ControlLoops
Mensagens Enviadas	IMC::SetServoPosition IMC::SetThrusterActuation
Parâmetros	“Maximum Thruster Output” (default: 1.0) “Maximum Thruster Differential” (default: 0.2) “Maximum Thruster Rate” (default: 0.0) “Low Speed Thruster Percentage” (default: 0.2) “High Speed Thruster Percentage” (default: 0.8) “High Speed Threshold” (default: 1.0 (nós)) “High Speed Hysteresis” (default: 0.1 (nós)) “Speed PID Gains” (default: nenhum) “Heading PID Gains” (default: nenhum)

Processo de Setup

Inicialização da Tarefa

Para além de se declararem e atribuírem os parâmetros externos às devidas variáveis, são subscritas as mensagens IMC de entrada. A velocidade desejada, e as atuações dos motores do ciclo anterior (será discutido mais à frente) são inicializados a 0, e a variável booleana de estado de “alta velocidade” é inicializada como “falso”.

onUpdateParameters

Se os ganhos dos PIDs forem alterados, ambos os PIDs são reiniciados (valores anteriores para efeito de cálculo de integrais são apagados) e são inicializados com os novos ganhos.

Inicializar o PID inclui limitar o seu output, que será determinado pelo parâmetro externo “Maximum Thruster Output” (p.ex. Se este parâmetro for “1” os PIDs serão limitados entre -1 e 1)

Processo Contínuo

Esta tarefa é reativa, o que quer dizer que, por si só, não exerce qualquer processamento sem estímulos externos. Esses estímulos correspondem às receções de mensagens IMC, elaborado mais à frente.

Main

A tarefa é reativa, pelo que o main apenas recebe as mensagens IMC externas.

Receção de Mensagens IMC

IMC::Abort

Verifica-se se a mensagem tem este sistema como destinatário, e, se assim for, reinicia-se os PID’s e enviam-se pontos de referência nulos, i.e., a embarcação fica parada. Note-se que, em princípio, este código não corre, pois, por norma, a mensagem IMC::Abort, leva a uma reinicialização do sistema. Contudo, no caso de isso não acontecer, este código serve de salvaguarda.

IMC::ControlLoops

Verifica-se se esta mensagem se refere a controlo de direção ou de velocidade (que são os relevantes para esta tarefa) e se esta mensagem é a de maior prioridade (no caso de múltiplas tarefas enviarem este tipo de mensagem). Se for, e se a mensagem for uma mensagem de ativação desses controlos, a tarefa é ativada, se já não o estiver, ou, caso a mensagem seja de desativação, a tarefa desativa-se, se já não estiver. No final, se a tarefa acabar por ficar desativada (sendo que já não estava previamente), então os PID’s são reiniciados e a embarcação fica imobilizada.

IMC::DesiredHeading e IMC::DesiredSpeed

Ao receber uma mensagem com nova referência de direção ou velocidade, a tarefa atualiza a variável local para corresponder à nova referência.

IMC::EstimatedState

É aqui que o controlo PID é realizado. Ao receber um novo estado do sistema são extraídos os campos de velocidade e direção, que são comparados com os valores de referência, sendo assim calculado o erro que é encaminhado para os controladores PID. Aquando da receção, é realizada a análise do tempo que passou entre esta receção e a receção anterior, para efeitos de cálculo de derivada. Cada controlador PID produz um resultado de controlo relativo à sua variável de controlo (direção ou velocidade). Estes outputs são processados da seguinte forma:

- Direção: se a velocidade for elevada (acima do threshold definido nos parâmtros) então uma percentagem significativa do valor de atuação será aplicada aos motores, sendo o

oposto aplicado aos lemes. Caso contrário será aplicada uma percentagem inferior aos motores, sendo favorecida a utilização dos lemes. Posteriormente, é verificado que o valor de controlo dos motores não é excessivamente alto (definido pelo parâmetro “Maximum Thruster Differential”) e, se for, o valor é reduzido para corresponder a esse limite. Finalmente, o valor dos lemes é normalizado para se encontrar entre 0 e 1 (que corresponde ao Duty Cycle a ser aplicado)

- Motores: com a componente diferencial dos motores calculada a partir do ponto anterior é calculada a componente de velocidade (i.e. a componente comum a ambos os motores, não a diferencial). Esta componente é combinada com a diferencial (o valor diferencial é somado para o motor a bombordo, e subtraído para o motor a estibordo). Se algum destes valores ultrapassar o limite de atuação máximo, definido pelo parâmetro “Maximum Thruster Output”, então o valor diferencial é reforçado, ou seja, o controlo de direção é favorecido ao de velocidade (exemplo: controlo de velocidade resulta em “1”, controlo de direção resulta em “0.2”, que cumulativamente significa que o motor a estibordo irá atuar “0.8” e o a bombordo “1.2” que está acima do valor default de “1”. Os valores são então reduzidos para “0.6” e “1” respetivamente para manter a diferença de “0.4” original, bem como o limite de atuação “1”).

Com os valores de atuação calculados é feito um último passo de processamento opcional: se o parâmetro “Maximum Thruster Rate” for definido como algo diferente de “0”, então os motores são forçados a não variar a sua atuação repentinamente (quer em menos atuação, quer na aceleração), evitando sinais de atuação que sejam altamente irregulares (ex. Estar imóvel, e, de repente, acelerar a potência máxima). A derivada é calculada pelo tempo entre a última mensagem recebida e esta mensagem, bem como a diferença entre a última atuação feita e esta atuação, sendo a “última atuação” atualizada no processo.
Finalmente, os valores calculados de atuação são enviados como IMC::SetServoPosition (para os lemes) e IMC::SetThrusterActuation (para os motores). Ambas estas mensagens estão entre 0 e 1. As mensagens possuem um campo de id que é “1” para o motor a bombordo, e “2” para o motor a estibordo. Os lemes também têm um id de “1”.

Debugging

O nível de debug utilizado é de “spew” (debug detalhado) e envia as seguintes mensagens:

- Notificação de receção de novo estado e valores dos erros calculados (direção e velocidade)
- Valor de atuação resultante, entre 0 e 1, para o motor a estibordo, motor a bombordo e lemes

Teste Unitário

Para testar esta componente isoladamente foi feita outra tarefa (“Navigation/Waypoints”) que simula, por um lado, o sensor, como por outro, o controlo de missão que envia a referência de direção e velocidade. Esta tarefa é muito simples, sendo periódica, e tem o seguinte comportamento:

1. Envia uma direção desejada de 90º (este) e velocidade de 3 nós
2. Nas próximas 10 iterações envia estados do sistema que se aproximam linearmente da posição e orientação desejada (note-se que isto é independente do controlo PID, seja qual for o seu comportamento, esta tarefa vai sempre produzir os mesmos valores).
3. Quando se atinge a velocidade e direção desejadas é enviado um novo par de valores de referência: -90º (oeste), 5 nós
4. Repete-se o ponto 2
5. Após atingir a velocidade e direção desejadas a tarefa limita-se a enviar permanentemente novos estados de erro 0 (isto é, mantém-se na direção e velocidade desejadas)

Teste de Integração

Para testar esta componente integrada com os elementos base do sistema, utilizou-se um simulador que gera valores de posição, velocidade e orientação com base nos valores de atuação que esta tarefa envia (IMC::SetThrusterActuation). Desta forma, enviaram-se manobras do tipo “Ir para…” para fazer um teste simplista, como prova de conceito.

Resultado

O controlador PID agia como esperado, direcionando o motor e os lemes conforme planeado: a velocidades baixas os motores mantém-se relativamente perto de 0.5 (50% duty cycle, ou seja, parados) e os lemes direcionam-se para a direção esperada. A altas velocidades o sistema apresenta maior atuação do lado dos motores, sendo os lemes relativamente inutilizados. Um problema detetado é que os lemes, em ambiente de simulação, não trazem qualquer vantagem, sendo difícil avaliar o algoritmo nesse aspeto.

Controlo de Altitude

Tarefa de supervisão do sistema. Quando os controladores de direção e/ou velocidade do sistema são ativados, a tarefa monitoriza a altitude do sistema em relação ao fundo do mar, podendo interromper a manobra atual, e potencialmente oferecer a possibilidade de executar uma manobra de correção. São definidos três parâmetros: a altitude mínima a que o sistema é considerado como estando em altitudes seguras; uma margem de altitude em que é permitida à embarcação executar uma manobra de correção; uma margem de altitude em que se considera que a embarcação ainda não voltou a uma altitude segura.

Informação Genérica

Nome da tarefa	MinAltitude
Descrição	Recebe altitude da embarcação em questão e avalia, de acordo com um parâmetro definido para a aplicação, se a distância ao fundo do mar é demasiado reduzida ou não. Se a distância for demasiado reduzida, a embarcação para de se mover.
Localização	src/Supervisors/MinAltitude/
Mensagens Recebidas	IMC::EstimatedState IMC::Abort IMC::ControlLoops
Mensagens Enviadas	IMC::Abort
Parâmetros	“Minimum Acceptable Altitude” (default: 4.0) “Minimum Altitude Margin” (default: 0.1) “Good Altitude Margin” (default: 0.1

Processo de Setup

Inicialização da Tarefa

Para além de se declararem e atribuírem os parâmetros externos às devidas variáveis, são subscritas as mensagens IMC de entrada. A altitude atual é inicializada a infinito (máximo valor possível). Assume-se que, inicialmente, o sistema não está a uma altitude demasiado reduzida.

onUpdateParameters

Se qualquer um dos parâmetros (altitude mínima admissível, margem de altitude mínima, e margem para se considerar uma boa altitude) for alterado, a altitude atual é imediatamente avaliada, de forma a ter em consideração os novos parâmetros.

Processo Contínuo

Esta tarefa é reativa, o que quer dizer que, por si só, não exerce qualquer processamento sem estímulos externos. Esses estímulos correspondem às receções de mensagens IMC, elaborado mais à frente.

Main

A tarefa é reativa, pelo que o main apenas recebe as mensagens IMC externas.

Receção de Mensagens IMC

IMC::EstimatedState

Ao receber a mensagem do Estado da embarcação, o campo de “altitude” é avaliado e comparado com a referência de altitude mínima admissível. Se a altitude for inferior ao limite, é enviada uma mensagem IMC::Abort.

Se o sistema estiver a uma altitude segura, para além de IMC::Abort, a tarefa comunica uma mensagem de erro que menciona a possibilidade de executar uma manobra de correção (uma manobra que desloque o sistema para uma zona de altitude segura). Durante esta manobra, existe uma margem extra de altitude em que o sistema é permitido navegar (definido pelo parâmetro “Minimum Altitude Margin”). Nesta manobra, quando a altitude for igual à altitude mínima aceitável, mais a margem definida (“Good Altitude Margin”) considera-se que o sistema está de volta a uma altitude aceitável. Se, nesta manobra, a altitude atingir valores inferiores à altitude mínima tendo em conta a margem concedida, o sistema envia outro IMC::Abort, e, desta vez, a embarcação irá permanecer imóvel, não permitindo manobras de correção.

IMC::ControlLoops

Para evitar que sejam enviados IMC::Abort desnecessariamente, esta tarefa é ativa apenas quando os controladores de direção ou velocidade estiverem ativos. Assim, evita-se que sejam enviados IMC::Abort de forma permanente quando a embarcação está imóvel/à deriva.

Debugging

O nível de debug utilizado é de “spew” (debug detalhado) e envia as altitudes atuais do sistema aquando de eventos significativos, nomeadamente:

- Altitude mínima aceitável ultrapassada, permitindo manobra de correção.
- Altitude mínima mais margem ultrapassada, sistema deixa de ficar operacional, até se encontrar numa altitude aceitável.
- Altitude mínima mais margem de recuperação restabelecida, sistema volta ao estado de funcionamento normal.

Teste Unitário

Para testar esta componente isoladamente, enviaram-se valores fixos de IMC::EstimatedState, de forma a forçar a embarcação a percorrer todas as possíveis sequências de eventos, bem como IMC::ControlLoops para ativar a tarefa:

1. Envio de uma altitude aceitável, seguida de uma altitude abaixo da aceitável, mas dentro da margem de manobra de correção, seguida de uma altitude demasiado reduzida (o sistema primeiro bloqueia, permitindo manobra de correção, e, posteriormente, bloqueia sem possibilidade de correção).
2. Envio de uma altitude aceitável, seguido de uma altitude extremamente reduzida (o sistema bloqueia de imediato, sem possibilidade de correção).
3. Envio de uma altitude extremamente reduzida, aquando da inicialização do sistema (mesmo resultado que o ponto anterior).
4. Envio de uma altitude aceitável, seguida de uma altitude abaixo da aceitável, seguida de uma altitude aceitável, mas abaixo da margem definida para se considerar que o sistema está a uma altitude aceitável novamente, seguida de uma altitude grande o suficiente para se considerar que o sistema está de volta a uma altitude aceitável (o sistema primeiro bloqueia, permitindo manobra de correção, e, posteriormente volta ao normal).

Teste de Integração

A tarefa foi adicionada ao ficheiro de configuração da embarcação e correu-se uma simulação que gera valores de altitude dependendo das coordenadas em que se encontra o sistema. Havendo conhecimento prévio da batimetria do mapa, realizaram-se os seguintes testes:

- Envio de manobra em direção a um ponto com águas demasiado superficiais (altitude baixa). Posterior envio de manobra de correção de forma a que a embarcação volte para trás (o sistema primeiro bloqueia, permitindo manobra de correção, e, posteriormente, bloqueia sem possibilidade de correção).
- Envio de manobra em direção a um ponto com águas demasiado superficiais (altitude baixa). Posterior envio de manobra de correção de forma a que a embarcação prossiga para a zona de reduzida altitude (o sistema primeiro bloqueia, permitindo manobra de correção, e, posteriormente volta ao normal).

Resultado

Dos resultados dos testes unitários e de integração, pode-se ter um bom grau de confiança de que a tarefa funciona. Note-se que esta tarefa assume que é possível obter a altitude a partir da mensagem IMC::EstimatedState, algo que não acontece no nosso sistema. Por esse motivo, esta tarefa apenas é testável por simulação.