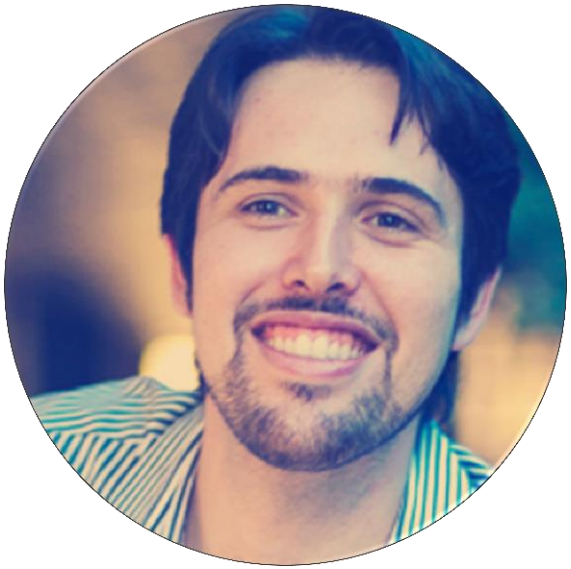
Python Não Usual

Pilotando Petroleiros

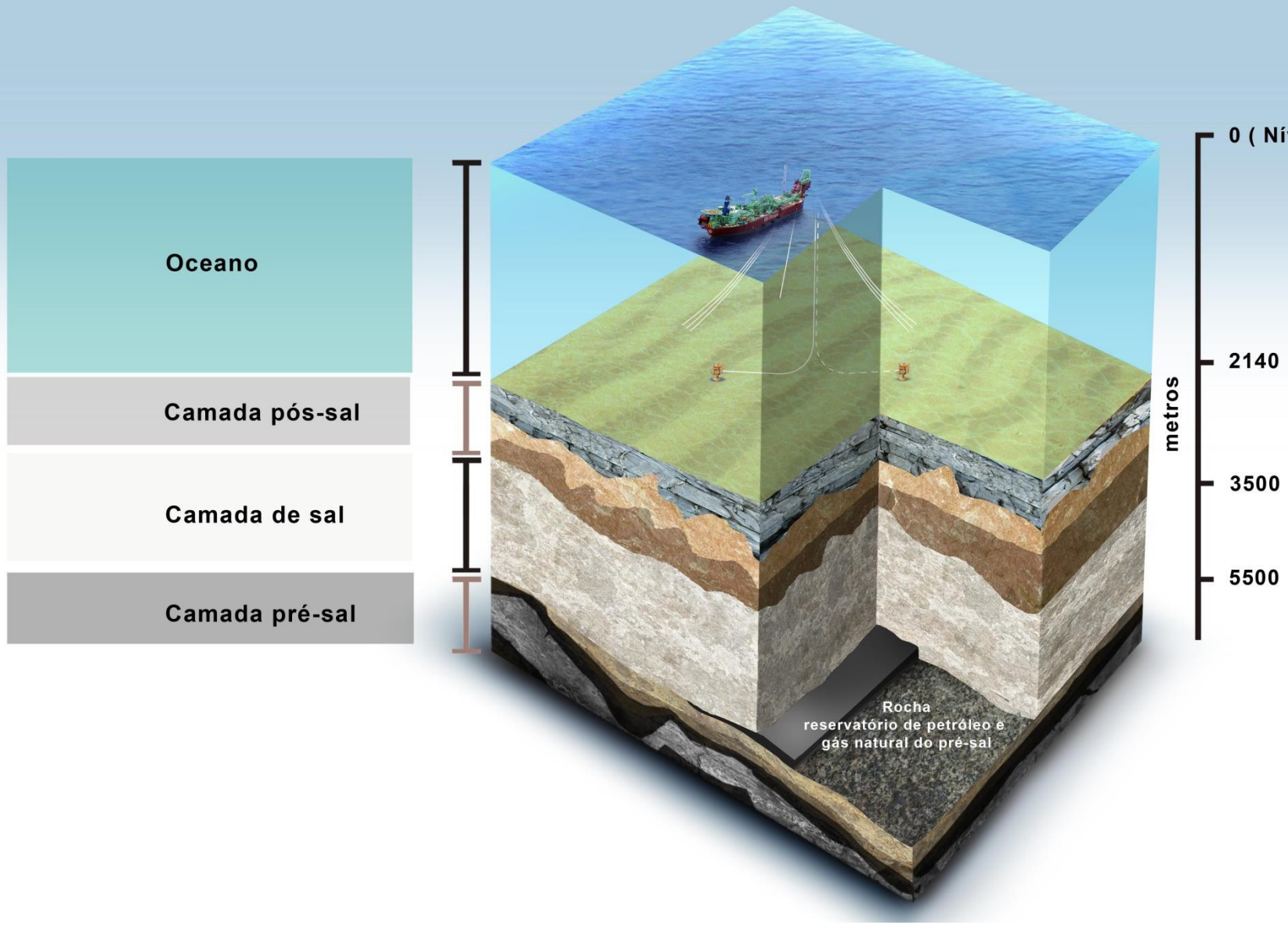
That's me, ~~Mario~~ Rafael!



- Mecatrônica (2010) e Mestrado em IA (2013)
- Fundador da Tegris (2010)
- Fundador do FieldLink (2014)
- Diretor do Escritório de Software da FDTE (2014)

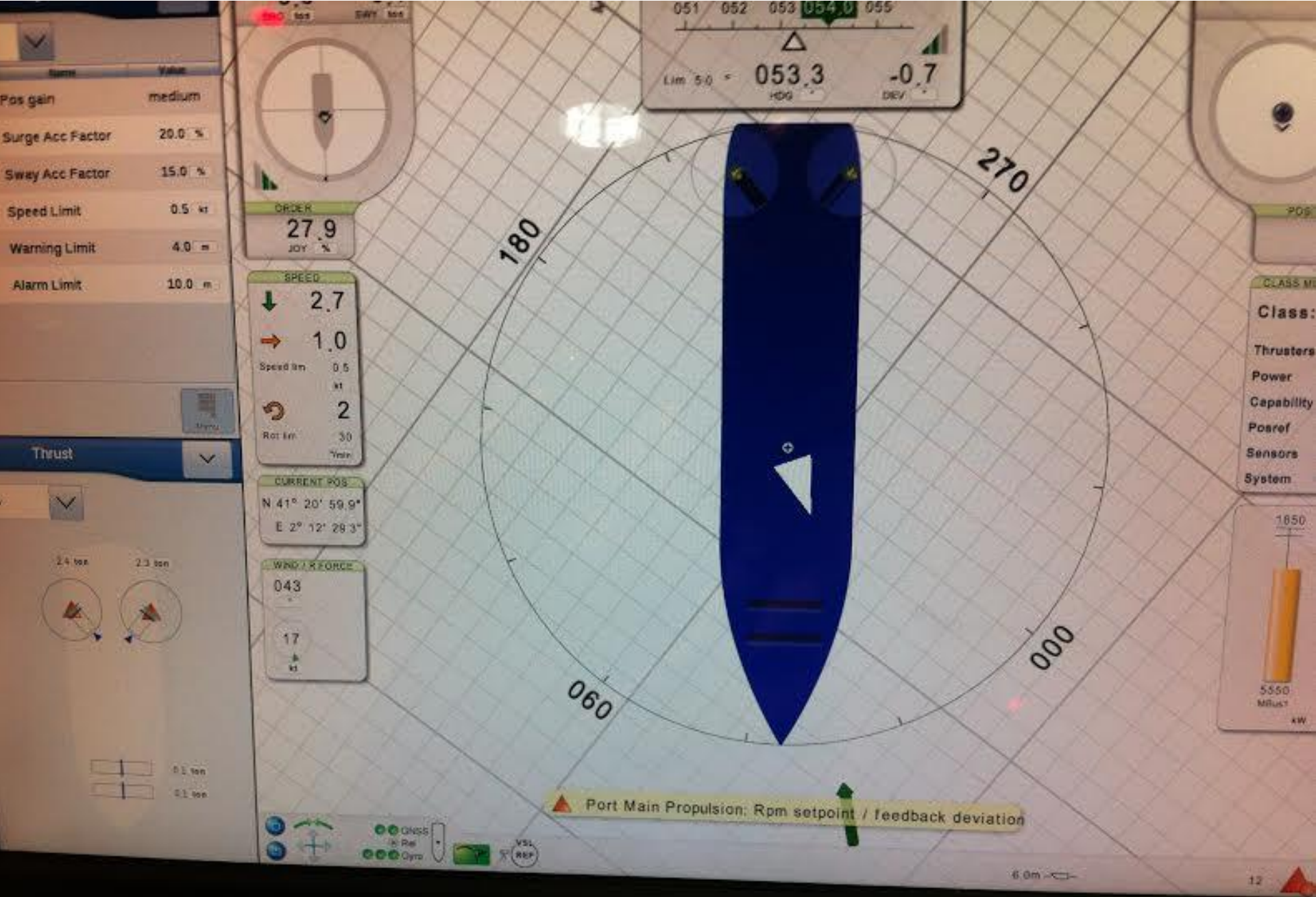
Twitter: @rafgoncalves

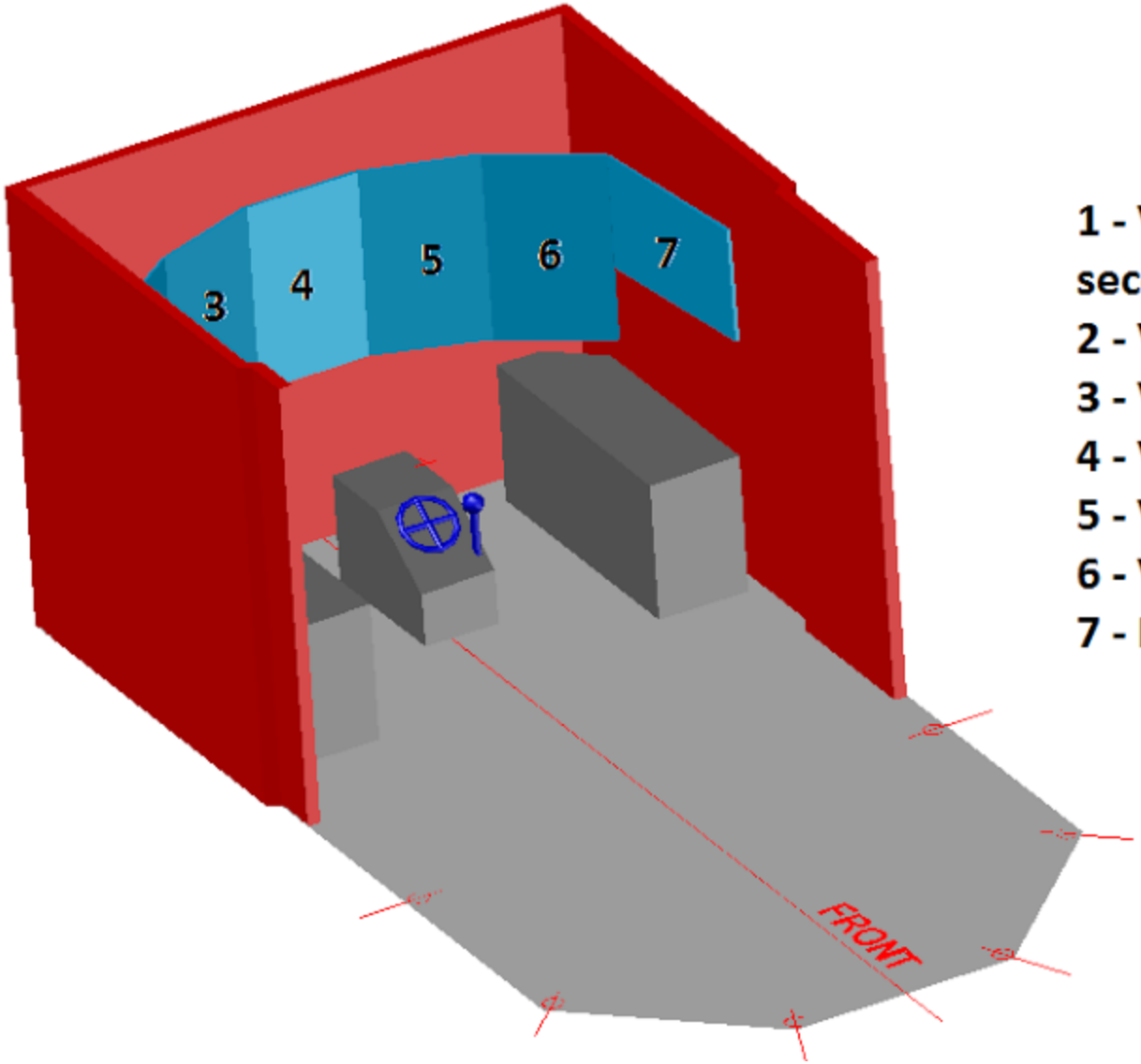
A produção no Pré-sal




ENTRADA
220
VOLTS


Rolls-Royce


Pos gain
medium
Surge Acc Factor
20.0 %
Sway Acc Factor
15.0 %
Speed Limit
0.5 kt
Warning Limit
4.0 m
Alarm Limit
10.0 m
Thrust
051 052 053 054.0 055
Lim 5.0
053.3
HDG
-0.7
DEV
ORDER
27.9
JOY %
SPEED
2.7
1.0
Speed lim
0.5
kt
2
Rot lim
30
°/min
CURRENT POS
N 41° 20' 59.9"
E 2° 12' 29.3"
WIND / K FORCE
043
17
kt
180
270
060
000
Class:
Thrusters
Power
Capability
Posref
Sensors
System
1650
5550
kW
Port Main Propulsion: Rpm setpoint / feedback deviation
GNSS
Rel
Gyro
VSL
REF
6.0m
12



1 - Visual 1 (from the secondary bridge)
2 - Visual 4
3 - Visual 5
4 - Visual 6
5 - Visual 7
6 - Visual 8
7 - No computer yet

Briefing Inicial

“Na verdade o projeto é simples. Nós vamos usar o cluster de simulação do Marin, então a função do integrador vai ser só decidir quem está controlando o simulador e enviar os dados para o cluster. Nada de mais.”

As Primeiras Surpresas

- A RRM trabalha com % de alocação, a Marin com RPM
- A RRM utiliza uma extensão proprietária do NMEA 0183
- O Marin trabalha com UDP, a RRM trabalha com CAN sobre serial
- Existem instrumentos que não são da RRM e que devem receber dados

Subindo o Nível

- Padronização de todo o projeto para utilização de NMEA 0183
- Definir uma DSL para especificação de novas mensagens
- Definir um tipo base e um gerador de testes

```
168
169 class RMC(BaseNMEA):
170     _template = '{talker_id:s}RMC,{utc:f},{status:s},{latitude:f},{latitude_NS:s},{longitude:f},' \
171                 '{longitude_EW:s},{SOG:f},{COG:f},{date:f},{magnetic_var:f},{magnetic_var_EW:s},{mode:s}'
172
173
174 class ROT(BaseNMEA):
175     _template = '{talker_id:s}ROT,{rate_of_turn:f},{status:s}'
176
177
178 class VBW(BaseNMEA):
179     _template = '{talker_id:s}VBW,{longitudinal_water_speed:f},{transverse_water_speed:f},' \
180                 '{status_water_speed:s},{longitudinal_ground_speed:f},{transverse_ground_speed:f},' \
181                 '{status_ground_speed:s},{stern_transverse_water_speed:f},{status_stern_water_speed:s},' \
182                 '{stern_transverse_ground_speed:f},{status_stern_ground_speed:s}'
183
184
185 class MWV(BaseNMEA):
186     _template = '{talker_id:s}MWV,{wind_angle:f},{relative_true:s},{wind_speed:f},{wind_unit:s},{status:s}'
187
188
189 class RSA(BaseNMEA):
190     _template = '{talker_id:s}RSA,{starboard_rudder_sensor:f},{starboard_rudder_sensor_status:s},{port_rudder_sensor:f},' \
191                 '{port_rudder_sensor_status:s}'
192
193
194 class RPM(BaseNMEA):
195     _template = '{talker_id:s}RPM,{source:s},{id_number:d},{rpm:f},{pitch:f},{status:s}'
196
197
198 class ZDA(BaseNMEA):
199     _template = '{talker_id:s}ZDA,{utf:f},{day:d},{month:d},{year:d},{local_hours:d},{local_minutes:d}'
200
201
202 class HDT(BaseNMEA):
203     _template = '{talker_id:s}HDT,{heading:f},{status:s}'
204
205
206 class PRC(BaseNMEA):
207     _template = '{talker_id:s}PRC,{lever_position:f},{status:s},{RPM_value:f},{RPM_mode:s},{pitch_value:f},' \
208                 '{pitch_mode:s},{operating_location:s},{engine_number:d}'
209
210
211 class CUR(BaseNMEA):
212     _template = '{talker_id:s}CUR,{validity:s},,,,{current_direction:f},{direction_reference:s},'\
213                 '{current_speed:f},,,{heading_reference:s},{speed_reference:s}'
214
215
```

```
297
298 class TestRSA(TestNMEAMessage):
299     sample_str = '$PCRSA,-35.0,A,0.0,V*5F'
300     sample_data = {'talker_id': 'PC','starboard_rudder_sensor':-35.0,'starboard_rudder_sensor_status':'A',
301                    'port_rudder_sensor':0.0, 'port_rudder_sensor_status':'V'}
302
303     def setUp(self):
304         self.nmea_obj = RSA()
305
306 class TestRPM(TestNMEAMessage):
307     sample_str = '$PCRPM,E,1,56.1,5.6,A*74'
308     sample_data = {'talker_id':'PC','source':'E','id_number':1,'rpm':56.1,'pitch':5.6,'status':'A'}
309
310     def setUp(self):
311         self.nmea_obj = RPM()
312
313 class TestZDA(TestNMEAMessage):
314     sample_str = '$GPZDA,123900.00,1,12,2011,12,39*56'
315     sample_data = {'talker_id':'GP','utf':123900.00,'day':1,'month':12,'year': 2011,
316                    'local_hours':12,'local_minutes':39}
317
318     def setUp(self):
319         self.nmea_obj = ZDA()
320
321 class TestHDT(TestNMEAMessage):
322     sample_str = '$HEHDT,56.9,T*15'
323     sample_data = {'talker_id': 'HE', 'heading': 56.9, 'status':'T'}
324
325     def setUp(self):
326         self.nmea_obj = HDT()
327
328 class TestCUR(TestNMEAMessage):
329     sample_str = '$GPCUR,A,,,,90.0,T,57.0,,,T,W*62'
330     sample_data = {'talker_id': 'GP','validity':'A','current_direction':90.0,'direction_reference':'T',
331                    'current_speed':57.0,'heading_reference':'T','speed_reference':'W'}
332
333     def setUp(self):
334         self.nmea_obj = CUR()
```

Ladeira Abaixo

- Na verdade, a RRM não trabalha com CAN, mas UDP
- Os instrumentos que haviam sido esquecidos são CAN e NMEA2000
- O DP só permite realizar operações depois que o navio passa em um teste de *circuit breakers*

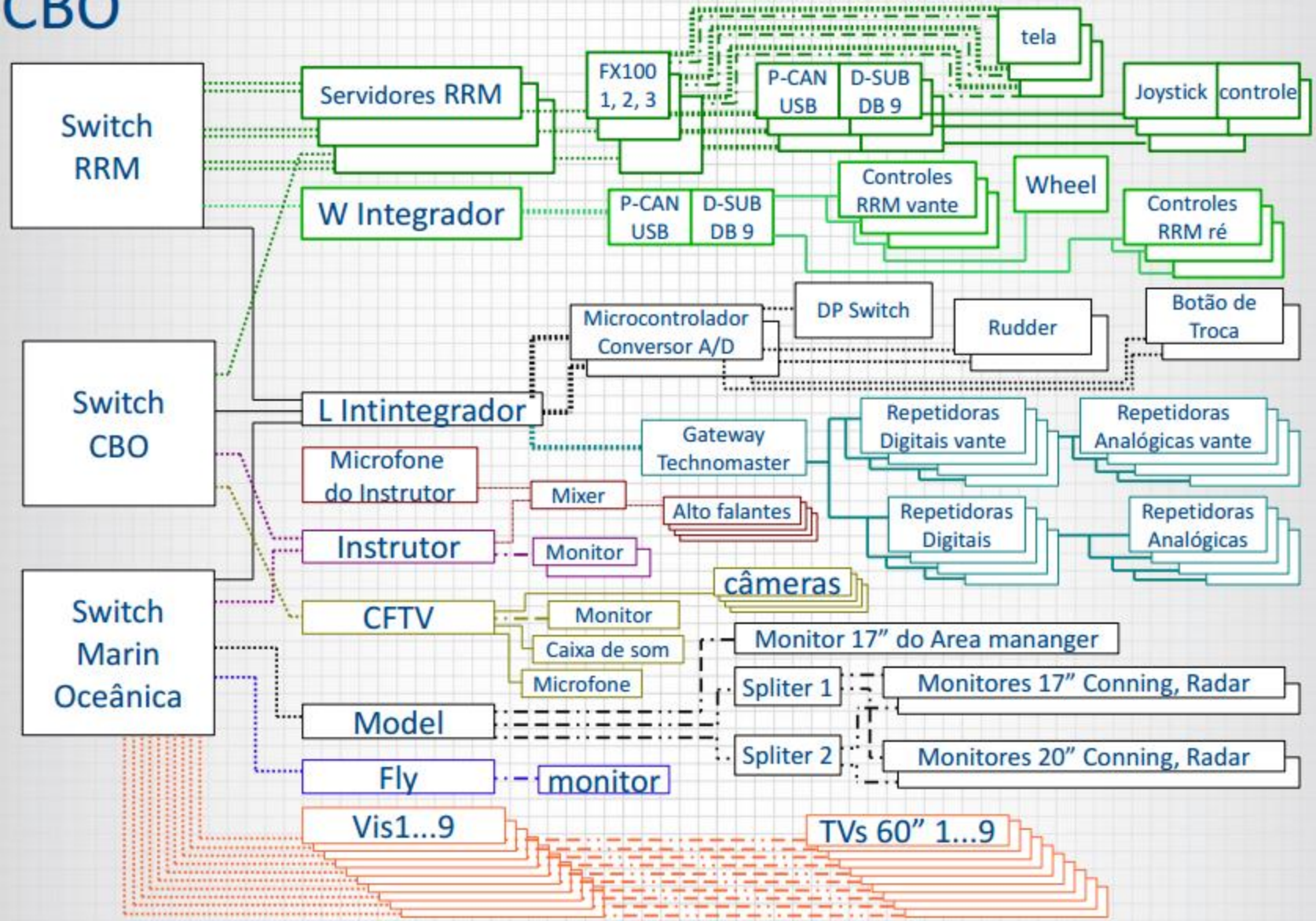
Tudo Pode Piorar

- Não basta ser UDP, na verdade a RRM utiliza um socket diferente para cada instrumento
- O DP e o PosCon precisa de fallback individual para controle manual
- O controle manual será feito com outro conjunto de manetes RRM

Parem o Mundo

- Assim, a arquitetura projetada para rodar com 4 threads passou a ter mais de 100...
- Como os instrumentos não se movimentavam suavemente, aumentou-se a frequência...
- Percebemos um comportamento estranho...

CBO


Switch RRM
Servidores RRM
FX100 1, 2, 3
P-CAN USB
D-SUB DB 9
tela
Joystick
controle
W Integrador
P-CAN USB
D-SUB DB 9
Controles RRM vante
Wheel
Controles RRM ré
Switch CBO
Microcontrolador Conversor A/D
DP Switch
Rudder
Botão de Troca
L Intintegrador
Gateway Technomaster
Repetidoras Digitais vante
Repetidoras Analógicas vante
Microfone do Instrutor
Mixer
Alto falantes
Repetidoras Digitais
Repetidoras Analógicas
Instrutor
Monitor
Switch Marin Oceânica
câmeras
CFTV
Monitor
Caixa de som
Microfone
Monitor 17" do Area mananger
Spliter 1
Monitores 17" Conning, Radar
Model
Spliter 2
Monitores 20" Conning, Radar
Fly
monitor
Vis1...9
TVs 60" 1...9

Uma Nova Esperança

- Utilizando programação assíncrona reescrever o código e eliminar os custos de switch
- TornadoWeb ou Twisted?
- Aproveitar o refatoring para “passar o código a limpo”

Comunicação de Baixo Nível

- O Tornado não possui suporte a UDP ou RS232...

```python
class BaseNMEAIO():
    _default_events = IOLoop.NONE

    def __init__(self, conn_parameters, name=None):
        self._conn_parameters = None
        self._stream = None
        self._current_events = IOLoop.NONE
        self._is_handler_active = False
        self._io_loop = IOLoop.instance()
        self.__configure(conn_parameters)
        self.name = name

    def __del__(self):
        # Makes sure the event_loop wont keep it hanging around
        if self._is_handler_active:
            self._remove_handler()

    def __configure(self, conn_parameters):
        self._stream = self._obtain_stream(conn_parameters)
        self._conn_parameters = conn_parameters
        self._add_handler(self._default_events)

    def _add_handler(self, events):
        self._io_loop.add_handler(self._stream.fileno(), self._handle_events, events)
        self._is_handler_active = True
        self._current_events = events

    def _remove_handler(self):
        self._io_loop.remove_handler(self._stream.fileno())
        self._is_handler_active = False
        self._current_events = IOLoop.NONE

    def _obtain_stream(self, conn_parameters):
        raise NotImplementedError

    # These parameters are required by the ioloop_add_handler
    def _handle_events(self, fd, events):
        raise NotImplementedError

```

```python
class BaseNMEAListener(BaseNMEAIO):
    _default_events = IOLoop.READ

    def __init__(self, *args, **kwargs):
        BaseNMEAIO.__init__(self, *args, **kwargs)
        self._specific_init()

    def _specific_init(self):
        self._nmea_factory = NMEAMessageFactory()
        self._read_callbacks = []

    def _recv(self):
        raise NotImplementedError

    def _handle_events(self, fd, events):
        msg = self._recv()

        try:
            msg_obj = self._nmea_factory.produce(msg.decode())
            for callback, types in self._read_callbacks:
                is_registered_type_for_this_callback = type(msg_obj) in types
                if is_registered_type_for_this_callback or not types:
                    # Schedule the callbacks for the next loop iteration
                    self._io_loop.add_callback_from_signal(callback, msg_obj)

        except TypeError as e:
            # There is no registered type for this message, discard it
            pass

        except UnicodeDecodeError as e:
            print('[WARN] Illegal character, could not resolve to UTF-8')

    def register_message_types(self, *args):
        self._nmea_factory.register(*args)

    def register_callback(self, callback, type_list=()):
        """
        :param callback: A function that will be invoked upon the event
        :param *types: Message types on which this callback will match. Note that the msg types must be
               already registered or an assertion fail will trigger.
        """
        assert callable(callback)
        for type_ in type_list:
            assert self._nmea_factory.is_registered(type_)
        self._read_callbacks.append((stack_context.wrap(callback), type_list))


```

```python
class BaseNMEAWriter(BaseNMEAIO):
    _default_events = IOLoop.WRITE

    def __init__(self, *args, **kwargs):
        buffer_size = kwargs.pop('buffer_size', 64)
        BaseNMEAIO.__init__(self, *args, **kwargs)
        self._specific_init(buffer_size)

    def __unsubscribe_from_write_event(self):
        events = xor(self._current_events, IOLoop.WRITE)
        if events == IOLoop.NONE:
            self._remove_handler()
        else:
            self._io_loop.update_handler(self._stream.fileno(), events)
            self._current_events = events

    def __subscribe_to_write_event(self):
        self._current_events |= IOLoop.WRITE

        if self._is_handler_active:
            self._io_loop.update_handler(self._stream.fileno(), self._current_events)
        else:
            self._add_handler(self._current_events)

    def _specific_init(self, buffer_size):
        self._register = Registrator()
        self._write_callbacks = []
        self._output_buffer = deque(maxlen=buffer_size)

    def _handle_events(self, fd, events):
        if len(self._output_buffer) > 0:
            msg_obj = self._output_buffer.popleft()
            self._send(msg_obj.to_bytestring())
            for callback, types in self._write_callbacks:
                is_registered_type_for_this_callback = type(msg_obj) in types
                if is_registered_type_for_this_callback or not types:
                    # Schedule the callbacks for the next loop iteration
                    self._io_loop.add_callback_from_signal(callback, msg_obj)
        else:
            # There is no more messages on the output buffer, we wont watch out for
            # writing anymore.
            self.__unsubscribe_from_write_event()

    def _send(self, bytes_):
        # Does the real sending, must be implemented in the device specific subclasses
        raise NotImplemented
```

```
238
239 class BaseUDPNMEAListener(BaseNMEAListener):
240     def _recv(self):
241         return self._stream.recv(1024)
242
243
244 class BaseUDPNMEAWriter(BaseNMEAWriter):
245     def _send(self, bytes_):
246         try:
247             self._stream.send(bytes_)
248         except socket.error:
249             print("[ERROR] Error on %s, sending %s" % (self.name, bytes_))
250
251 class UnicastUDPNMEAListener(BaseUDPNMEAListener):
252     def _obtain_stream(self, conn_parameters):
253         """
254         :param conn_parameters: is expected to be a tuple (interface, port) to be
255                 bound
256         """
257         interface_ip, port = conn_parameters
258         return UDPUtils.unicast_listener(interface_ip, port)
259
260 class UnicastUDPNMEAWriter(BaseUDPNMEAWriter):
261     def _obtain_stream(self, conn_parameters):
262         """
263         :param conn_parameters: is expected to be a tuple (destiny_ip, port) to be
264                 bound
265         """
266         destiny_ip, port = conn_parameters
267         return UDPUtils.unicast_writer(destiny_ip, port)
268
269 class MulticastUDPNMEAListener(BaseUDPNMEAListener):
270     def _obtain_stream(self, conn_parameters):
271         """
272         :param conn_parameters: is expected to be a tuple (interface, port, mcast_group) to
273                 connect
274         """
275         interface_ip, port, multicast_group = conn_parameters
276         return UDPUtils.new_multicast_sock(interface_ip, port, multicast_group)
277
```

Orientação a Eventos

- Cada mensagem que passa pelo integrador é um evento
- Cada evento pode acionar uma série de *callbacks*
- Os *callbacks* podem alterar estados e/ou disparar novas mensagens

```
'simulator':
[
    {
        'name': 'SIMULATOR',
        'io_class': MulticastUDPNMEAListener,
        'conn_parameters': ('192.168.10.100', 30100, '224.10.10.100'),
        'listen_types': [MWV, ROT, HDT, XN, TRD, VTG, ZDA, GST, GGA, CYS, SHP, PSAT, CUR, VBW, RMC],
        'callbacks': [
            {'function': save_azimuth_info_to_context_memory_during_wheel('PROPELLER_1', 'PROPELLER_2'), 'm

            # DP SPECIFIC
            {'function': relay_on_talker_id('dp', anemometer_table), 'matched_types': [MWV]},
            {'function': relay_on_talker_id('dp', gyro_table), 'matched_types': [ROT, HDT]},
            {'function': relay_on_talker_id('dp', vru_table), 'matched_types': [XN]},
            {'function': relay_on_talker_id('dp', gnss_table), 'matched_types': [VTG, ZDA, GST]},
            {'function': fill_fake_gga_data('dp', gnss_table), 'matched_types': [GGA]},
            {'function': thruster_data_to_rrm('dp', 'dp'), 'matched_types': [TRD]},
            {'function': correct_cys_messages('dp', 'CYSCAN'), 'matched_types': [CYS]},

            # POSCON SPECIFIC
            {'function': relay_on_talker_id('poscon', gyro_table), 'matched_types': [ROT, HDT]},
            {'function': relay_on_talker_id('poscon', gnss_table), 'matched_types': [VTG, ZDA, GST]},
            {'function': fill_fake_gga_data('poscon', gnss_table), 'matched_types': [GGA]},
            {'function': thruster_data_to_rrm('poscon', 'poscon'), 'matched_types': [TRD]},
            # {'function': print, 'matched_types': [SHP]},
            #
            # TECHNOMASTER
            # Handles [TRC, ETL, PRC, ROR, ZDA, RPM, RSA, MWV, VBW, CUR, ROT, PSAT, RMC]
            {
                'function': relay_with_talker_id_based_on_msg_type('instruments', 'TECHNOMASTER', technoma
                'matched_types': [ZDA, ROT, MWV, PSAT, CUR, VBW, RMC, TRD]
            },
        ]
    },
],
```

Closures

- Se utilizar de closures e “funções que retornam funções” para definir regras de alto nível.

```
from common.messages import PRC
from common.util import MathUtil
from cbo.context import Context
from rollsroyce.messages import THRF, THRLF, STGF, SGLF
from marin.messages import TRC, RUS, SBV
from cbo.messages import ARDF


def relay_on_talker_id(adapter_name, relay_table):

    def _fcn(msg_obj):
        adapter = Context().get_writer(adapter_name)

        if adapter.writing:
            writer = relay_table.get(msg_obj.get('talker_id'), None)
            if writer is not None:
                try:
                    adapter.send_to_writer(writer, msg_obj)

                except KeyError:
                    print('[ERROR] Adapter %s has no writer %s' % (adapter, writer))
        else:
            print('[WARNING] Adapter %s isn\'t write-ready, discarding %s' % (adapter, msg_o

    return _fcn


def relay_with_talker_id_based_on_msg_type(adapter_name, writer_name, table):
    def _fcn(msg_obj):
        adapter = Context().get_writer(adapter_name)

        if adapter.writing:
            if msg_obj.has_field('talker_id'):
```

Máquinas de Estado

- Cada componente é modelado como uma máquina de estados
  - Thrusters
  - Comando de visualização
  - Controle de passadiço
- A responsabilidade de validar a transição de estados é do próprio componente

```
def get_rudder(self, rudder_id):
    if rudder_id not in self._rudders.keys():
        def on_before_to_auto(e):
            return self.get_operation_mode().current in ('dp', 'poscon')

        fsm = Fysom({'initial': 'manual',
                     'events': [
                         {'name': 'to_auto', 'src': ['manual', 'manual_both', 'wheel'], 'dst': 'auto'},
                         {'name': 'to_manual', 'src': ['manual_both', 'wheel', 'auto'], 'dst': 'manual'},
                     ],
                     'callbacks': {
                         'onbeforeto_auto': on_before_to_auto,
                     }
        })

        self._rudders[rudder_id] = fsm

    return self._rudders[rudder_id]


def get_operation_mode(self):
    if self._operation_mode_instance is None:

        def deactivate_auto_controls(e):
            devices = chain(self._thrusters.values(), self._rudders.values())
            for d in devices:
                if d.current == 'auto':
                    d.to_manual()

        def force_bridge_to_aft_and_deactivate_auto(e):
            self.get_bridge_switch()._force_to_aft()
            deactivate_auto_controls(e)
```

Contexto e Injeção de Dependência

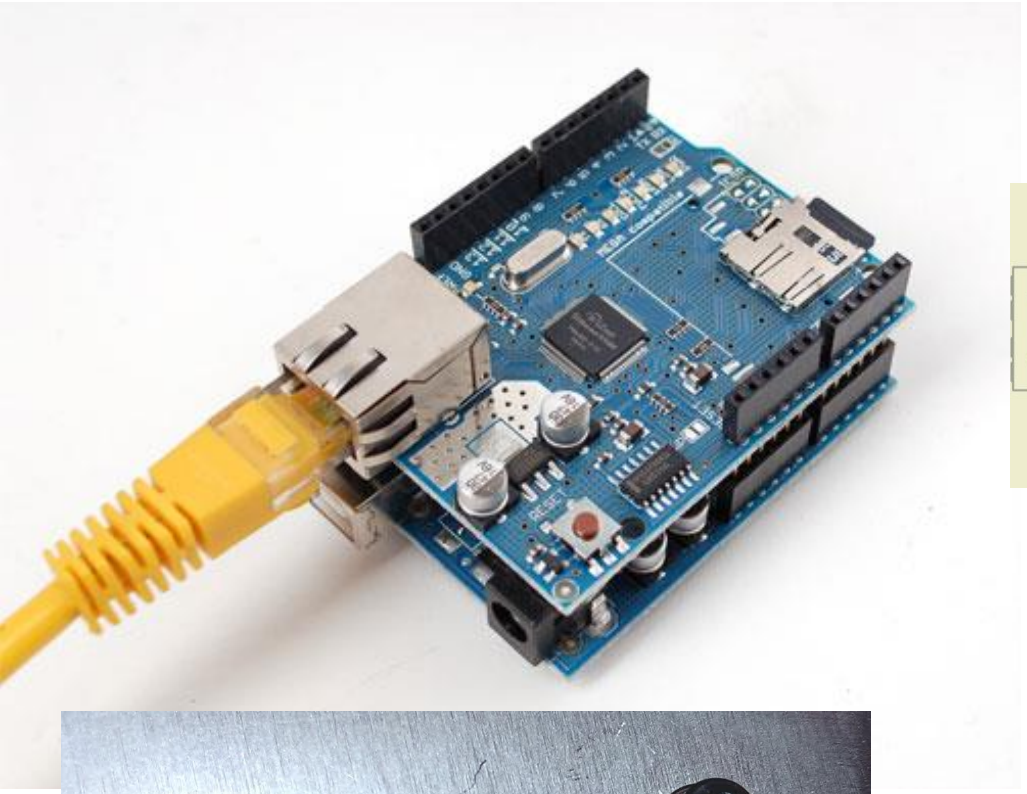
- Diferenciar declarativamente cada configuração dos navios
- Um gerenciador de contexto que lê os arquivos de configuração e constrói todas as máquinas de estado e árvores de *callbacks*

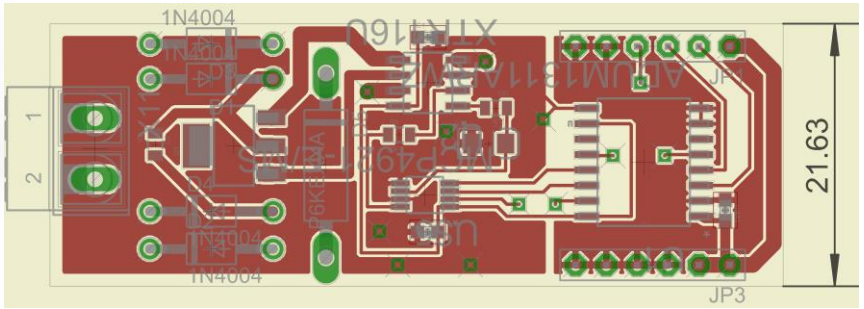
```python
import importlib
from common.util import Singleton
from common.adapters import RoutedListenerAdapter, RoutedWriterAdapter, RoutedDuplexAdapter
from cbo.state_machines import StateMachines
from settings import SHIPS

class Context(metaclass=Singleton):

    def __init__(self):
        self._listeners = {}
        self._writers = {}
        self.memory = {}
        self.ship_id = 0

    def set(self, k, v):
        self.memory[k] = v

    def get(self, k, default):
        return self.memory.get(k, default)

    def ship_id(self):
        return self._ship_module['ship_id']

    def load_ship(self, ship_id):
        self._load_ship(ship_id)
        self._sm = StateMachines()
        self._setup_writers()
        self._setup_listeners()
        self._set_up_duplexes()
        self.ship_id = ship_id

    def _load_ship(self, ship_id):
        self._ship_module = importlib.import_module(SHIPS[ship_id])

    def _setup_listeners(self):
        for name, config in self._ship_module.LISTENERS.items():
            self._listeners[name] = RoutedListenerAdapter(name, config)
            self._listeners[name].configure_listeners(keep_trying=True)

    def _setup_writers(self):
        for name, config in self._ship_module.WRITERS.items():
            self._writers[name] = RoutedWriterAdapter(name, config)
            self._writers[name].configure_writers(keep_trying=True)

```

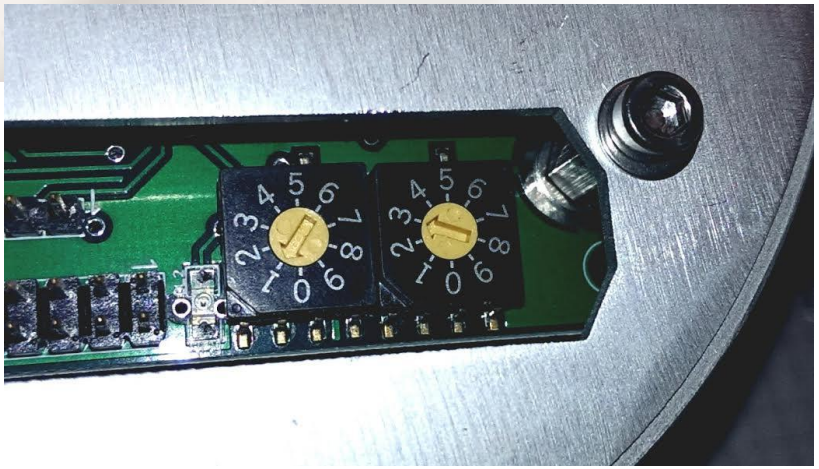
Novos Requisitos

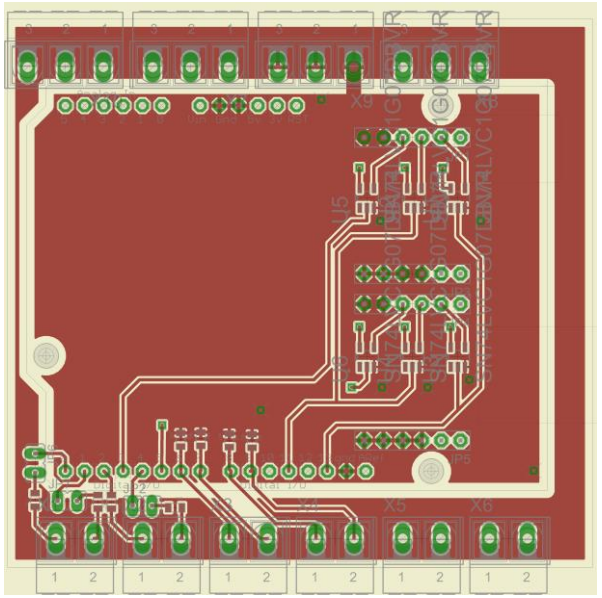
- Suporte a botoeiras multiuso
- Possibilidade de utilizar controles navais 100% analógicos
  - Potenciômetros
  - Contato seco
  - Loops de corrente
- Trocar automática de navios ao reiniciar a simulação no cluster Marin

Quem tem medo de hardware?





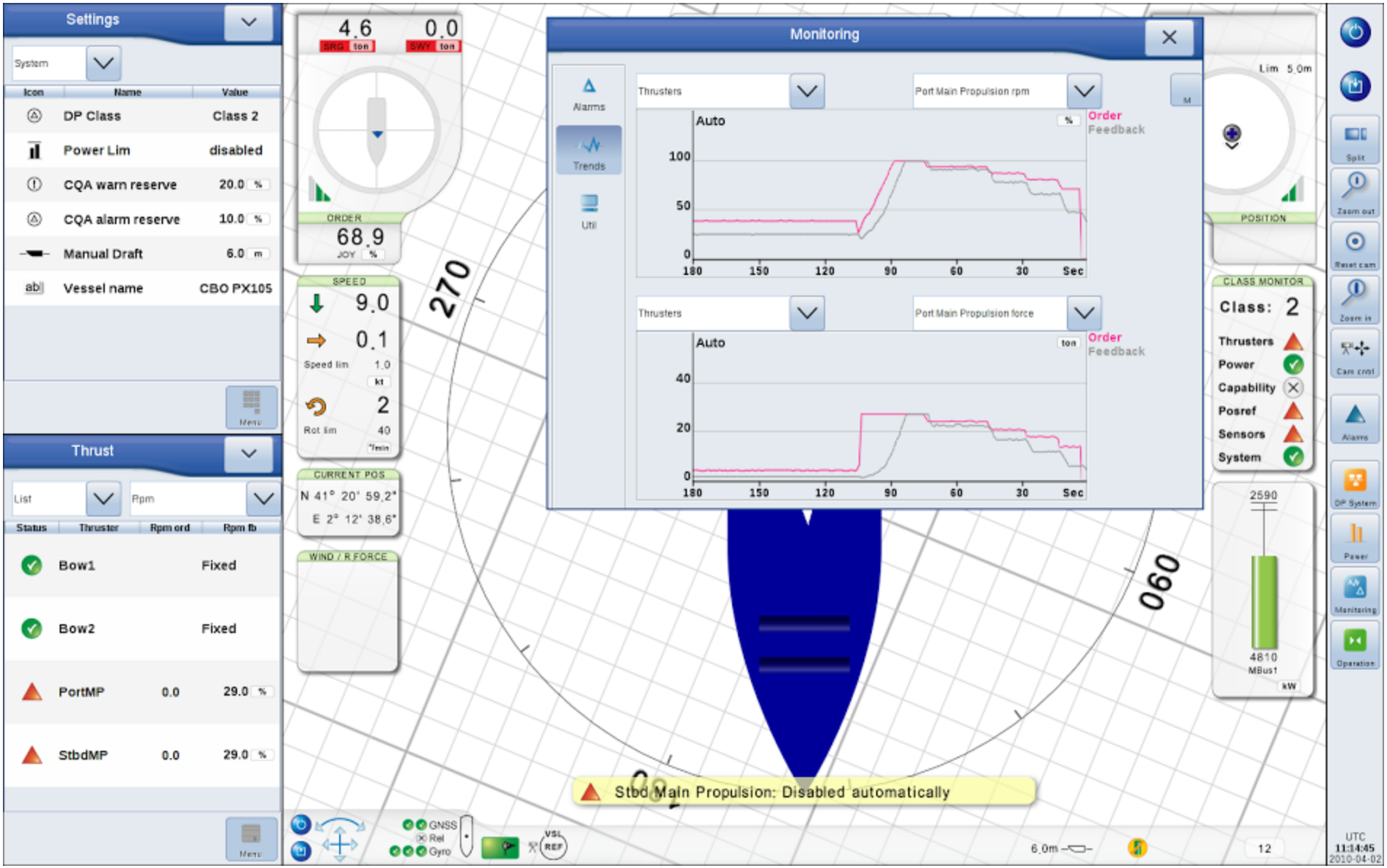


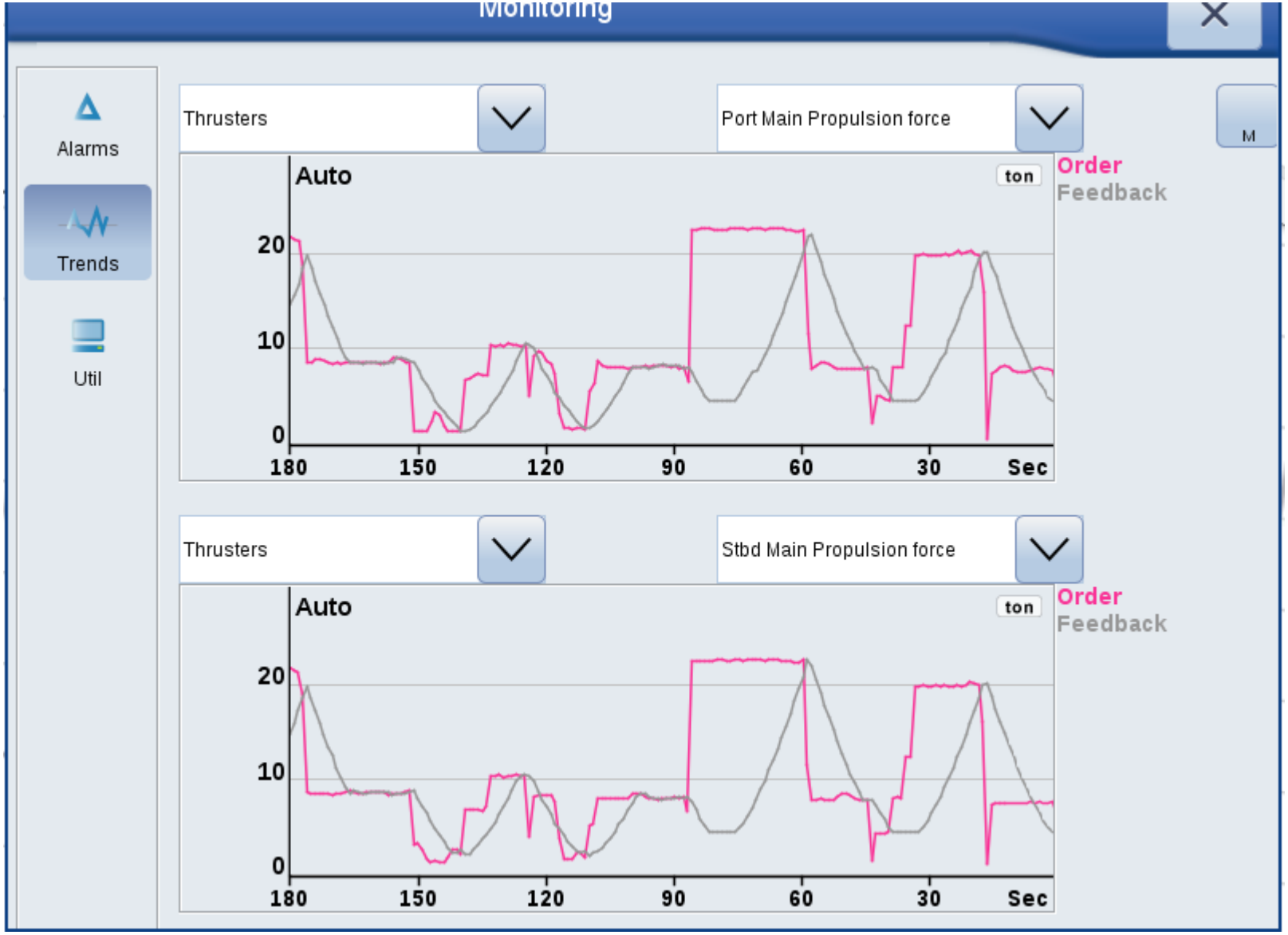


Jogando Sujo

- Trocar de navio significa refazer o objeto de contexto e todas as injeções de dependência
- Ao invés de fazer isso, podemos reiniciar a aplicação
- Com o Supervisord, isso é trivial!

O Último Teste


Settings
System
Icon
Name
Value
DP Class
Class 2
Power Lim
disabled
CQA warn reserve
20.0 %
CQA alarm reserve
10.0 %
Manual Draft
6.0 m
Vessel name
CBO PX105
Menu
Thrust
List
Rpm
Status
Thruster
Rpm ord
Rpm fb
Bow1
Fixed
Bow2
Fixed
PortMP
0.0
29.0 %
StbdMP
0.0
29.0 %
Menu
4.6
SRG ton
0.0
SWY ton
ORDER
68.9
JOY %
SPEED
9.0
0.1
Speed lim
1.0
kt
2
Rot lim
40
°/min
CURRENT POS
N 41° 20' 59.2"
E 2° 12' 38.6"
WIND / R.FORCE
Monitoring
Alarms
Trends
Util
Thrusters
Port Main Propulsion rpm
Auto
%
Order
Feedback
100
50
0
180
150
120
90
60
30
Sec
Thrusters
Port Main Propulsion force
Auto
ton
Order
Feedback
40
20
0
180
150
120
90
60
30
Sec
Lim 5.0m
POSITION
CLASS MONITOR
Class: 2
Thrusters
Power
Capability
Posref
Sensors
System
2590
4810
MBus1
kW
270
060
Stbd Main Propulsion: Disabled automatically
GNSS
Rel
Gyro
VSL
REF
6.0m
12
Split
Zoom out
Reset cam
Zoom in
Cam cntrl
Alarms
DP System
Power
Monitoring
Operation
UTC
11:14:45
2010-04-02


Monitoring
Alarms
Trends
Util
Thrusters
Port Main Propulsion force
M
Auto
ton
Order
Feedback
20
10
0
180
150
120
90
60
30
Sec
Thrusters
Stbd Main Propulsion force
Auto
ton
Order
Feedback
20
10
0
180
150
120
90
60
30
Sec


MARIN


Transporte de 70 milhões de
toneladas de carga por ano

Perguntas e Repostas

Muito Obrigado

- Quer trocar ideias?
  - Twitter, @rafconvalves
  - LinkedIn
  - E-Mail, rgoncalves@tegris.com.br