UNIVERSITÄT OSNABRÜCK

Vom Chaos zur Korrektheit

Zustände mit Minot gezielt verifizieren

Mittwoch, 19.11.2025

Christopher Sieh

Roboter haben Zustand.

Position, Planung, Fortschritt, Joints...

Stateful Nodes

ROS unterstützt Zustände mit Nodes.

Entwicklung komplexerer Systeme weiter herausfordernd.

Stateful Nodes

ROS unterstützt Zustände mit Nodes.

Entwicklung komplexerer Systeme weiter herausfordernd.

1. **Netzwerkprobleme** Alle QoS außer Reliable können Nachrichten (lokal) verlieren.
   → Netzwerk verändert Ergebnis

2. **Laufzeit** Node zu langsam für Sensordaten, dadurch Nachrichtenverlust.
   → Hardware verändert Ergebnis

3. **Fehlerbehebung** Iterationen schwerfällig. Tooling breit aber nicht gezielt.
   → Verifikation

Für Reproduzierbarkeit haben wir Bagfiles

Für Reproduzierbarkeit haben wir Bagfiles

Helfen die uns hier wirklich?

Bagfiles

Bagfiles

Netzwerkprobleme Publish mit QoS.

Bagfiles

- **Netzwerkprobleme** Publish mit QoS.

Bagfiles

-  **Netzwerkprobleme** Publish mit QoS.

  Laufzeit Zeit verlangsamen oder Start/Stopp. Weiterhin Publish & Forget.

Bagfiles

- **Netzwerkprobleme** Publish mit QoS.
- **Laufzeit** Zeit verlangsamen oder Start/Stopp. Weiterhin Publish & Forget.

Bagfiles

- **Netzwerkprobleme** Publish mit QoS.
- **Laufzeit** Zeit verlangsamen oder Start/Stopp. Weiterhin Publish & Forget.

Fehlerbehebung Bagfiles kürzen, Filter über CLI, manuell auslesen...

Bagfiles

- **Netzwerkprobleme** Publish mit QoS.
- **Laufzeit** Zeit verlangsamen oder Start/Stopp. Weiterhin Publish & Forget.
- **Fehlerbehebung** Bagfiles kürzen, Filter über CLI, manuell auslesen...

Bagfiles

- **Netzwerkprobleme** Publish mit QoS.
- **Laufzeit** Zeit verlangsamen oder Start/Stopp. Weiterhin Publish & Forget.
- **Fehlerbehebung** Bagfiles kürzen, Filter über CLI, manuell auslesen...

Ja, es gibt Wege. Alle bringen neue Fehlerquellen.

Bagfiles

- 🩹 **Netzwerkprobleme** Publish mit QoS.
- 🩹 **Laufzeit** Zeit verlangsamen oder Start/Stopp. Weiterhin Publish & Forget.
- 🩹 **Fehlerbehebung** Bagfiles kürzen, Filter über CLI, manuell auslesen...

Ja, es gibt Wege. Alle bringen neue Fehlerquellen. Daher haben wir 🩹.

Bagfiles

- 🩹 **Netzwerkprobleme** Publish mit QoS.
- 🩹 **Laufzeit** Zeit verlangsamen oder Start/Stopp. Weiterhin Publish & Forget.
- 🩹 **Fehlerbehebung** Bagfiles kürzen, Filter über CLI, manuell auslesen...

Ja, es gibt Wege. Alle bringen neue Fehlerquellen. Daher haben wir 🩹.

Wir wollen aber ✅.

Grüne Wiese

✅ **Netzwerkprobleme** QoS nur mit Opt-In.

Grüne Wiese

✅ **Netzwerkprobleme** QoS nur mit Opt-In.

✅ **Laufzeit** Asynchrone ⟶ Synchrone Kommunikation.

Grüne Wiese

✅ **Netzwerkprobleme** QoS nur mit Opt-In.

✅ **Laufzeit** Asynchrone ⟶ Synchrone Kommunikation.

✅ **Fehlerbehebung** Benutzerfreundliche, einheitliche, flexible Kontrolle über Input.

Minot

✅ **Netzwerkprobleme** QoS nur mit Opt-In.

✅ **Laufzeit** Asynchrone ⟶ Synchrone Kommunikation.

✅ **Fehlerbehebung** Benutzerfreundliche, einheitliche, flexible Kontrolle über Input.

⟶ **Minot**

Terminal Anwendung mit VSCode Integration und Sammlung von C bzw. Rust Bibliotheken.

Minots Ledge Light bei Boston 1978 fotografiert von Kevin Cole.

'Fehlerbehebung'

Bagfiles als Datenbank mit Cursor.

Bagfiles

Nutzerschnittstelle ist **Code**.

Bagfiles

Nutzerschnittstelle ist **Code**.

Integrierter Compiler für zielgerichtete Sprache.

Bagfiles

Nutzerschnittstelle ist **Code**.

Integrierter Compiler für zielgerichtete Sprache.

```
1   _bag.{
2     cloud.{
3       _topic = "/ouster/points"
4       _type = Cloud
5     }
6     imu.{
7       _topic = "/ouster/imu"
8       _type = Imu
9     }
10  }
11  _start_locked = false
```

Setze einige Filter.

```
13  reset! ./bagfile
14  pf! cloud 1
```

Lade die Bagfile
und spiele die **nächste** LiDAR Nachricht.

'Laufzeit'

Asynchrones Pub/Sub synchronisieren.

Minot-Loop

Node gibt vor, **wann** Bagfile publisht.

```
pf! [imu, cloud], cloud 1 tick_end
pf! cloud 1 tick_end
pf! imu 1 tick_end
```

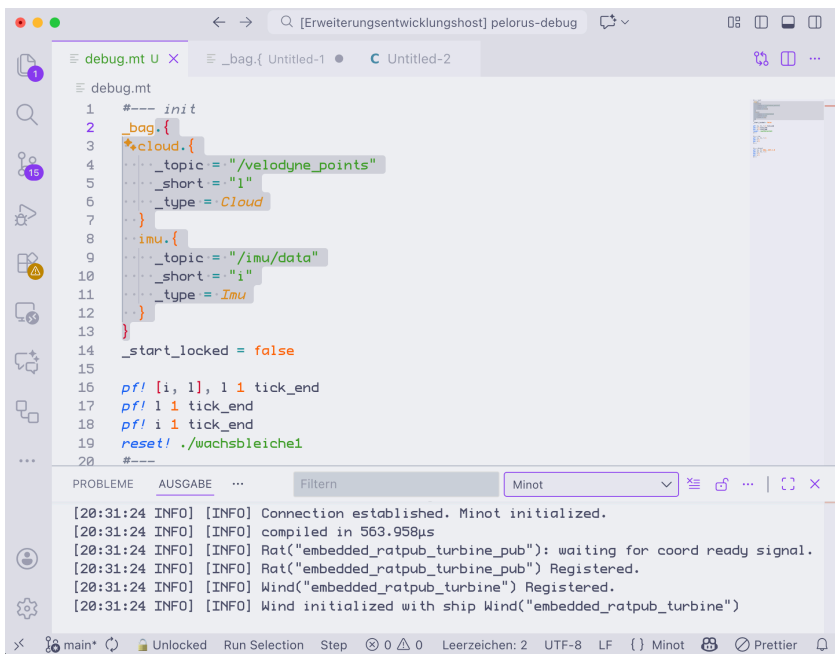
Registriere Abfragen bei **Auslösen** von „tick_end“.

```c
rat_init("my_lio", -1);

// your code here...

unsigned char* tick_end = NULL;
tick_end = malloc(sizeof(*tick_end));
rat_bacon_u8("tick_end", tick_end, 1, 1);
free(tick_end);

// your code here...
```

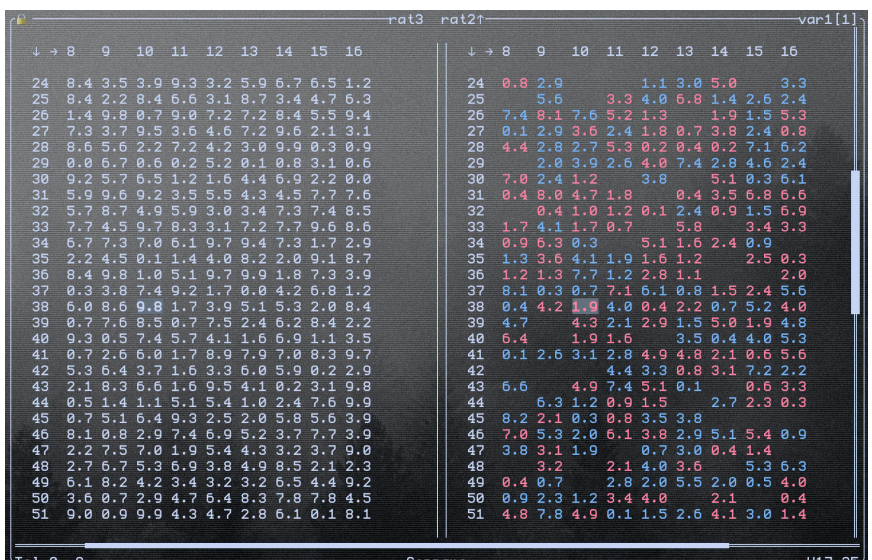
Löse die Abfragen und das Publishen aus (C).

Interface

Visual Studio Code oder Terminal



VSCode Erweiterung für Bagfile Query.



Terminal UI mit erweiterter Funktionalität,
hier Vergleich von Variablen.

Beispiel LIO (VSCode)

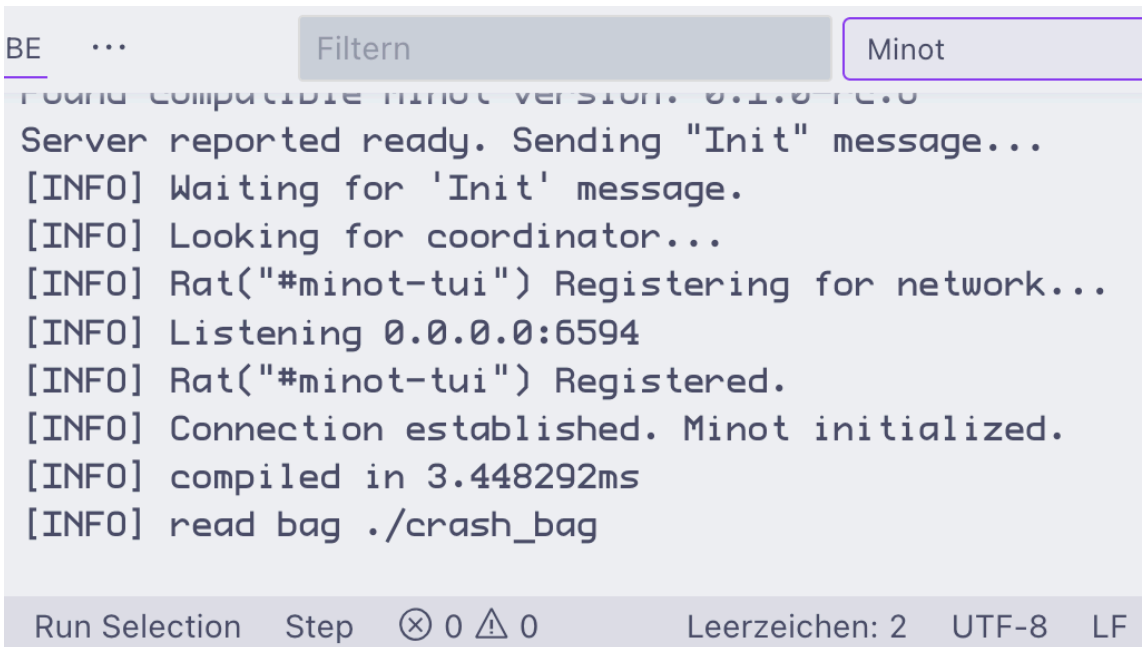
Intuitives, **selektives Ausführen** von Zeilen.

Der **Zustand wird gehalten** (vgl. Jupyter Notebook).

```
_bag.{
  cloud.{
    _topic = "/ouster/points"
    _type = Cloud
  }
  imu.{
    _topic = "/ouster/imu"
    _type = Imu
  }
}
_start_locked = false

reset! ./bagfile
pf! cloud 1
```

Beispiel LIO (VSCode)

Log bestätigt: Bagfile wurde gelesen.



Beispiel LIO (VSCode)

LIO wird über Minot Pub/Sub verbunden.

```
[INFO] Send channel closed, send_to_socket task exiting.
[INFO] Send channel closed, send_to_socket task exiting.
[INFO] Registering for topics ("pelorus_node", "/tf", Subscribe)
[INFO] Registering for topics ("pelorus_node", "/tf_static", Subscribe)
[INFO] Registering for topics ("pelorus_node", "/velodyne_points", Subscribe)
[INFO] Registering for topics ("pelorus_node", "/imu/data", Subscribe)
[INFO] Registering for topics ("pelorus_node", "registered", Publish)
[INFO] Registering for topics ("pelorus_node", "odom", Publish)
[INFO] Registering for topics ("pelorus_node", "path", Publish)
[INFO] Registering for topics ("pelorus_node", "imu_odom", Publish)
[INFO] Registering for topics ("pelorus_node", "imu_path", Publish)
```

Run Selection Step ⊗ 0 ⚠ 0 Leerzeichen: 2 UTF-8 LF {} Minot Prettier

Beispiel LIO (VSCode)

LIO wird über Minot Pub/Sub verbunden.

```
pf! [imu, cloud], cloud 1
pf! cloud 1
pf! imu 1
```

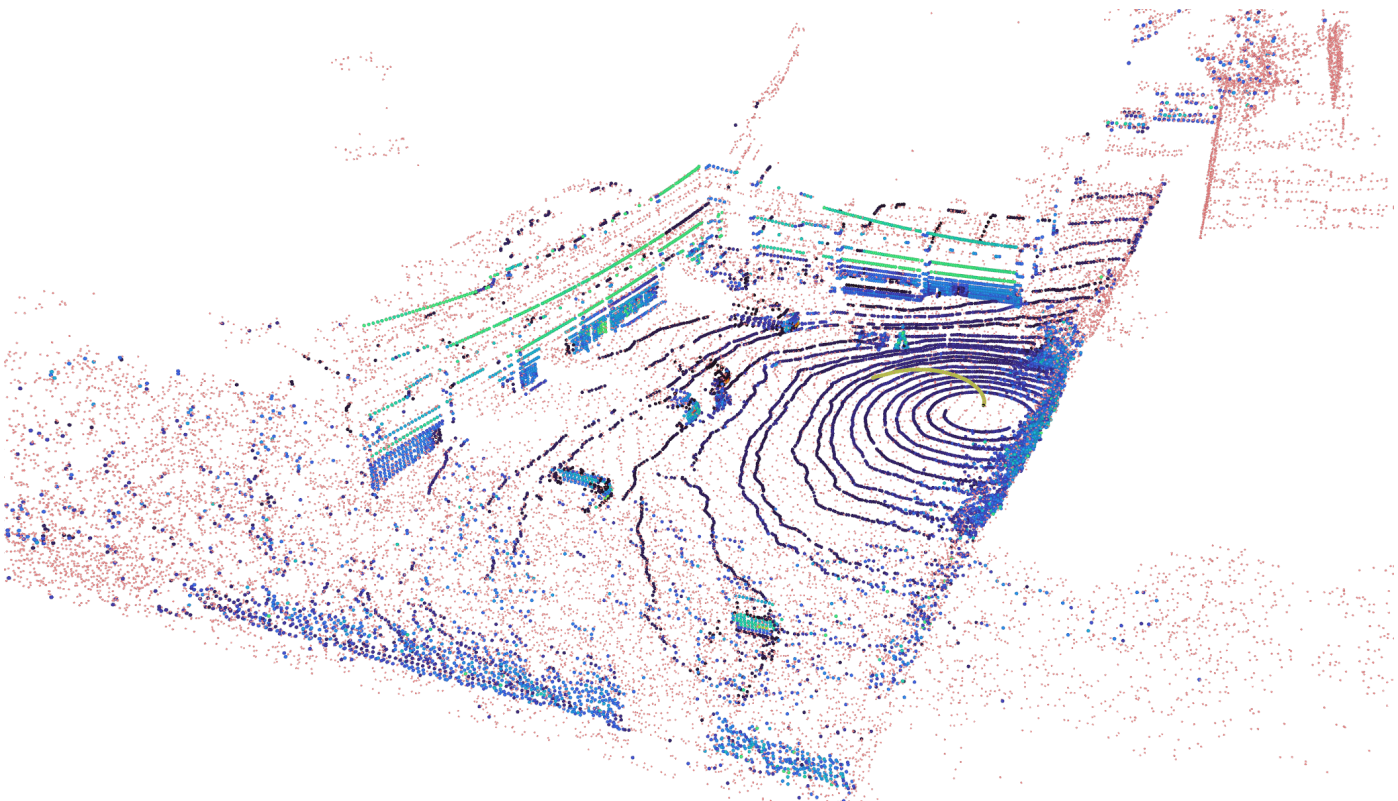
Stoße **Loop** an.

```
[23:50:53 INFO] [INFO] read bag ./crash_bag
[23:51:00 INFO] [INFO] compiled in 158.667µs
[23:51:00 INFO] [INFO] iterating on bagfile...
[23:51:00 INFO] [INFO] got 13 msgs in 6.137625ms
[23:51:00 INFO] [INFO] iterating on bagfile...
[23:51:00 INFO] [INFO] Registering for topics ("embedded_ratpub_turbine_pub", "/ouster/imu",
Publish)
[23:51:00 INFO] [INFO] got 1 msg in 14.040375ms
[23:51:00 INFO] [INFO] iterating on bagfile...
[23:51:00 INFO] [INFO] got 1 msg in 10.495875ms
[23:51:00 INFO] [INFO] Registering for topics ("embedded_ratpub_turbine_pub", "/ouster/
points", Publish)
```

Minot publisht zu LIO.

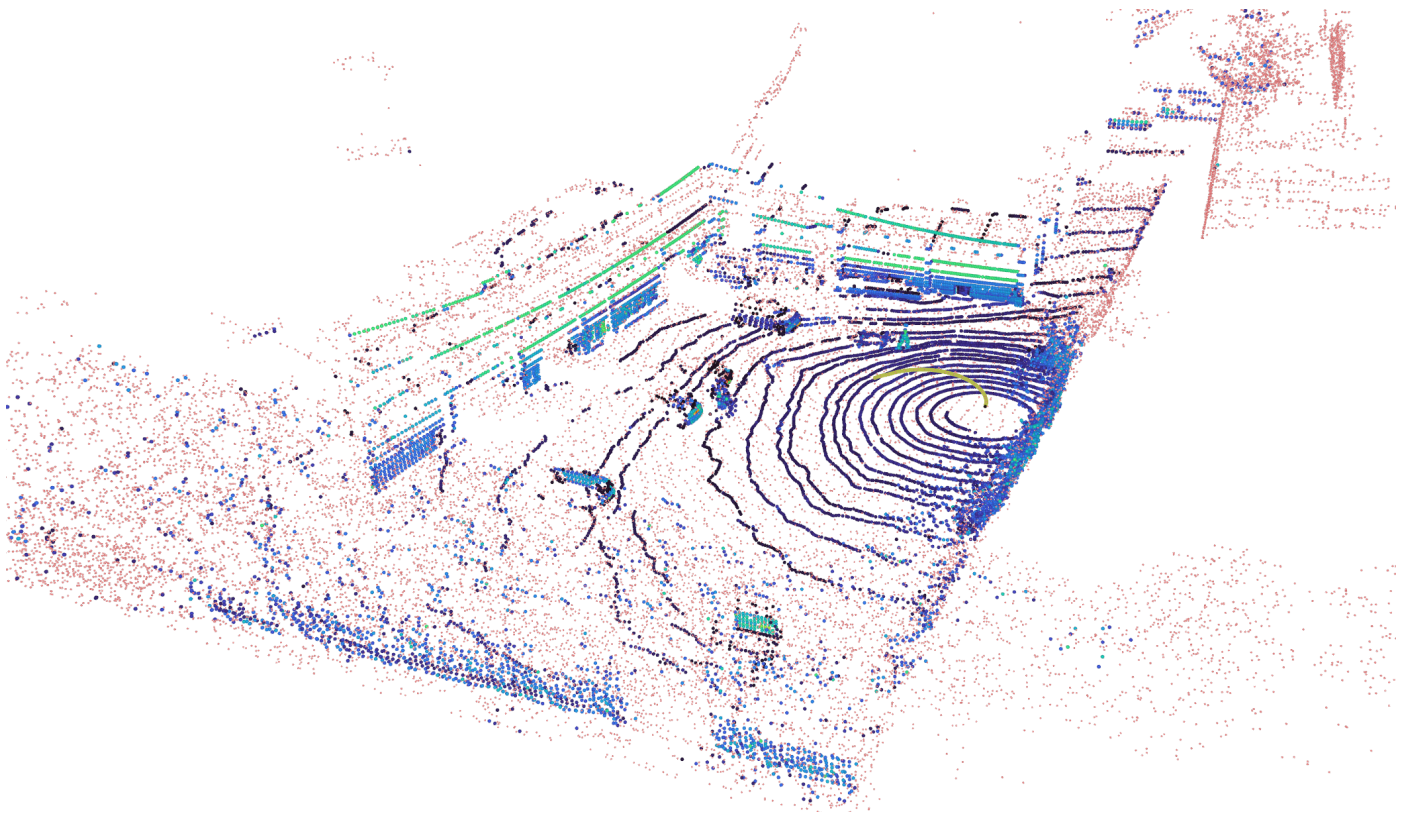
Beispiel LIO (VSCode)

LIO läuft.



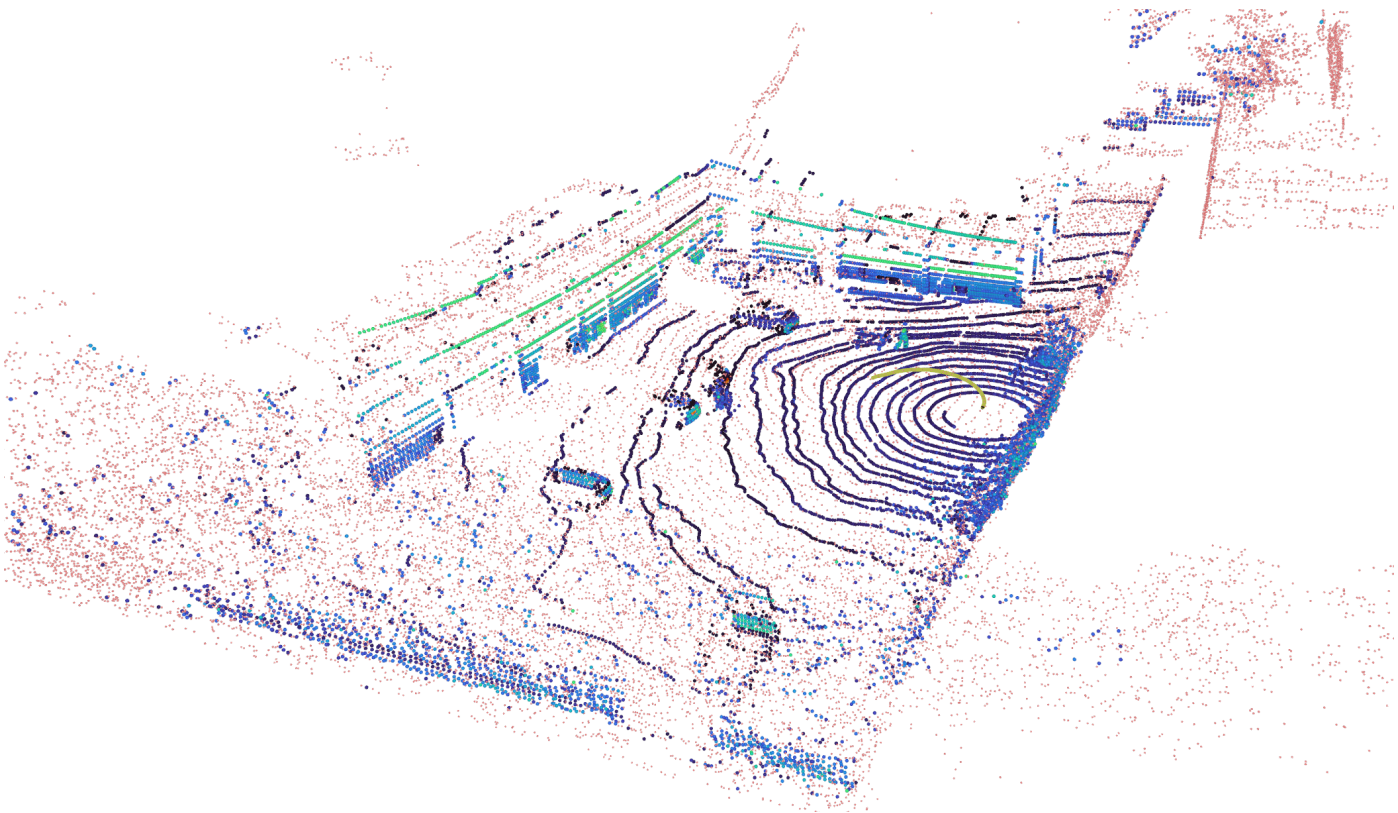
Beispiel LIO (VSCode)

LIO läuft.



Beispiel LIO (VSCode)

LIO läuft.



Und wie wird verifiziert?

Der Auslöser (im Beispiel ‚tick_end‘) ist technisch die Anfrage, ob die Variable ‚tick_end‘ synchronisiert werden soll.

Variablen Synchronisieren

Speicher übers Netzwerk **synchronisieren**.

```
# ...
reset! ./bagfile
# declare sharing rules
rule! (myvar
  # Send myvar from node1 to node2, overwriting node2
  node1 -> node2

  # node3 overwrite both variables of node1 and node2
  node3 <- node1, node2

  # Both nodes send their myvar to Minot where they
  # can be compared with each other
  blue == yellow

  # node1 sends it myvar to the Minot TUI for displaying it
  node1 -> LOG
)
pf! cloud 1 # ... continue normal execution
# ...
```

Variablen Synchronisieren

Ein **Prozess im Netzwerk** kann Werte zu einem Zeitpunkt synchron **testen**.

```
rule! (var
  lio -> e2e
)

rule! (var3
  lio -> e2e
)

rule! (var4
  lio -> e2e
)
```

Sende alle Zwischenergebnisse zum Testprozess.

```
ret = rat_bacon_u8("var", var, 1, 1);
if (ret != 0) {
  printf("Error while bacon var");
  return ret;
}
assert(*var == true);
free(var);
```

Asserte das Ergebnis.

Und Minot kann noch vieles mehr...

- Läuft überall: Windows, Linux, Mac, iOS, Android...
- Pub/Sub ohne QoS (nur Rust) – **‚Netzwerkprobleme‘**
- Gleichzeitig publishen nach ROS1, ROS2, Minot Pub/Sub (Ratpub)
- Komplett verteilte, modulare Architektur
- Open Source (MIT oder Apache-2) und dokumentiert
- Mehr **Szenarien gesucht**!

Informationen auf *https://uos.github.io/minot*

Danke!

Christopher Sieh – **chsieh@uos.de**
Wissenschaftlicher Mitarbeiter

AG Technische Informatik
Prof. Dr.-Ing. Mario Porrmann
https://inf.uos.de/ce

Universität Osnabrück

Institut für Informatik
Wachsbleiche 27
49090 Osnabrück